Development Of A Multi-Player WebSim In BT

BY MURTEZA SALEMI

Submitted to the Faculty of Engineering in
Partial Fulfillment of the Requirements for the Degree of

SIVILINGENIØR IN INFORMATION AND COMMUNICATION TECHNOLOGY

At Agder College (Norway)

Grimstad, June 1999

Contents

Executive Summary

This report investigates the development of multi-player WebSims using the Powersim Metro technology. A WebSim is a simulation of some phenomenon that can be played as a game on the Internet or Intranet. The WebSim front-end is the user interface to a simulation model that has been developed with a system dynamics modeling tool like Powersim Constructor.

The WebSim front-end is an interactive interface that allows a user to enter inputs to the underlying simulation model and start the simulation. The user can see the results of the simulation in form of time-graphs or time-tables.

WebSims can be single-player or multi-player games. This report is concerned with multi-player WebSims.

The two main issues that this report covers are:

- What is the process for development of a multi-player WebSim;
- How robust this technology is regarding working on different computer-platforms and browsers;

To answer these two questions, I have developed a multi-player WebSim called "OrgSim" and tested it in a number of configurations. This development was done as a collaboration with the Business Modelling Team at BT Laboratories, UK.

In OrgSim four players can participate in the game and compete with each other. OrgSim consists of two windows. One is a wizard window and the other one is the main window where the user will play. The wizard window collects some information from the user. Powersim Metro Session Manager allows for running several sessions of OrgSim. Therefore in the wizard window the user has the option to either create a new session or to join an existing running session. After gathering the other needed information in the wizard window (like the user's name and the session name), the main window is shown to the user.

In the main window the user will commence the play. The main window consists of several tab panels which time-graphs, time-tables, sliders, buttons, text boxes, labels and etc are placed on these tab panels. Objects like time-graphs and time-tables are used to show the results of the simulation and objects like text boxes and sliders enter the user inputs.

The main conclusions drawn from the research are as follows:

- Using Powersim Metro to develop a multi-player WebSim is not a complicated process and rather it follows a well defined process;
- Although Powersim Metro facilitates the development of a multi-player Websim, but considerable care and some ingenuity are needed in the development process;
- The challenge in developing a multi-player Websim (like OrgSim) lies primarily in implementing the WebSim front-end rather than in usage of the Powersim Metro tools;
- In general the multi-player simulator OrgSim works and is reasonably robust. However, some robustness and reliability problems remain which would require further development to address;
- OrgSim runs on both browsers IE 4.01 (or later versions) and NC 4.08 (or later versions);
- OrgSim front-end has an interactive interface that provides the player with both input and output facilities (input objects like text boxes and sliders to enter the inputs and output objects like time-graphs and time-table to show the results).

  OrgSim also provides a simple communication between the players ("hurry up" messages can be sent between the players);
- OrgSim has some undesired features that the cause for some of these features are not known for certain and further investigation and research are needed to clarify the problems;

Chapter 1 Introduction

1.1 Context

By using system dynamics methodology, dynamic and complex systems like the environment that surrounds an organization can be modeled and simulated.
BT (British Telecommunication plc) has developed such models in Powersim Constructor for 5 years to support Management and Marketing activities.
A team at BT labs has also had considerable experience with multi-player simulations (Turkey conference paper 1997).
Recently with the launch of Powersim Metro, the BT labs team has investigated the use of WebSims. The reason for trying WebSims is because of some interesting characters that WebSims provide.
Examples of these existing characters are, providing the user with an friendly interface to the system dynamics models to interact with effectively, can easily be browsed and downloaded to a local machine as an applet and at last the joy of playing these simulators (WebSims) that can be very entertaining for the users.

The subject of the following thesis was about to carry out some research on developing a multi-player WebSim, build up a multi-player WebSim, research on running the WebSim on different platforms and test the PsWebsim technology that is used for development of multi-player WebSims.
The research and development of the multi-player WebSim is done for British Telecommunication plc partially in Norway and partially in England.
The result of the research is the following report plus a multi-player WebSim called OrgSim.
The report is divided in 6 chapters which the process of the research and development of OrgSim are gradually covered through these chapters.

1.2 The Motivation For The Research

BT has developed a single user WebSim for one of its models (made in Powersim Constructor) using the WebSim technology and Borland JBuilder tool.
Because of the limited exercise and the difficulties with running the WebSim on different platforms, it was sensed that some research in this field could be worth to carry out. The research was planned to be done on multi-player WebSims. The two main reasons for choosing the multi-player WebSim were to gain some broader knowledge and experience on WebSim technology and because of BT's existing use of multi-player simulations.
In brief the main goals and objectives that BT wanted to gain from this research were as follows:

- What can be achieved by multi-player WebSims;
- The feasibility of the technology;
- What is the process of developing a multi-player simulator;
- How robust and reliable this technology is when run on different platforms;

In order to be able to answer the above issues, the multi-player simulator "OrgSim" was constructed. On other word, OrgSim is intended to be as a demonstrator to observe the extent of the achievement of the above objectives.

1.3 Thesis' Research Subject

The thesis subject is the development of a multi-user (player) simulator based on a model that was already made in Powersim Constructor by BT, research on the PsWebsim technology in making the multi-user simulators and research on deployment of the simulator on different platforms.

The objectives of the research were as follows:

- build and setup a multi-player simulator using Powersim Metro;
- research and documenting the process of developing the WebSim;
- research on how robust this technology is regarding working on different pc'platforms and browsers;

Some issues that are beyond the scope of this thesis and can be researched in the future if required are:

- research on how effective WebSim is compared to other war gaming implementations and configurations;
- research on the validity of the underlying model;

It should be noted that the reason for choosing OrgModel (the model that was developed by BT in Powersim constructor) was primarily because it was an existing simple model that could be adapted easily for WebSim implementation and not because it addressed any particular BT business issues.

In the next chapter we will see what WebSims exactly are and what the purposes are for having them.

Chapter 2 A Presentation Of WebSim Technology

2.1 Introduction to Java and Java Applet

Before presenting the WebSim technology in detail, some basic and general knowledge about Java is useful here.

Java is an object-oriented programming language developed by Sun Microsystems. This language is originally modeled after C++ and was designed to be small, portable, robust, and object-oriented. Java is specifically designed to be platform-independent. This means that programs written in Java can be compiled once and run on any machine that supports Java. In the other words the programs are capable of running on various computer systems without the necessity of being recompiled.

2.1.1 What is an applet?

An applet is an interactive program that runs inside a Web page when that page is displayed

by Java-capable browser, such as Netscape Navigator, Microsoft Internet Explorer, or Sun's HotJava browser.

Java applets are dynamic and interactive, so they expand the types of transactions that users can accomplish on the Web. After creating and compiling the Java applet, it is embedded in an HTML (Hypertext Markup Language) Web page and then publish the Web page on a Web site. If the user is accessing the Web site with a Java-enabled browser, the browser will download the embedded applet's binary code to the user's computer system and execute it.

There are some terms that I will be using often in this report and are worth explaining them here in this chapter. These terms are as follows in the next sections.

2.1.2 What is a class in Java?

A class is a template from which objects with similar aspects can be created. Classes embody all the features of a particular set of objects.

2.1.3 What is an object?

An object is an actual class instance. The class is the generic representation of an object and an instance is the concrete thing created from the instructions provided

by a class. We can think of a class as the architectural plans and the class instance as the actual building.

2.1.4 What is a method?

A method is a function (subroutine or procedure) defined inside a class that operates on instances of that class.

2.1.5 What is a constructor?

A constructor is a class method used to create instances of the class. Calling a constructor initializes the object and its variables, creates any other objects that the object needs, and generally performs any other operations the object needs in order to initialize itself.
A class can have several different constructors.

2.1.6 What is inheritance?

Inheritance is the mechanism that allows a new class to receive (inherit) the basis of its functionality from an existing class and build on that base by adding new functionality.
The class from which the current class is derived, is called a superclass. The superclass is above the current class in the class hierarchy. The class that is derived from the super class is called subclass. The subclass of the current superclass is lower in the class hierarchy.

2.1.7 What is an interface?

In Java each class can only have one superclass. The subclass inherits variables and methods from that superclass and all its superclasses. This single inheritance has advantages and drawbacks. The main drawback is that single inheritance can be somewhat restricting. This is especially true when you have similar behavior that needs to be duplicated across different branches of the class hierarchy. Java has solved this problem of shared behavior by introducing the concept of interfaces. An interface is a collection of method declarations without actual implementations. Although a single Java class can have only one superclass due to single inheritance, that class can implement any number of interfaces.

2.1.8 What is a package?

A package is a collection of related classes and interfaces. Packages are a way of relating certain classes and interfaces so that they can be referred to and imported as a group.

2.2 WebSim

WebSim (simulator) is a general concept for Java applet that can be run via a browser on the Internet. The WebSim front-end is the user interface to a simulation model that has been developed in Powersim Constructor.
System dynamics is used to simulate a reality that exists in a complex environment and consists of factors that are distributed in time and place and interact with each other rather directly or indirectly. The distribution of these factors over time and place and their dependencies on each other, all together are collected and placed in one single model.
WebSim is an interface to this model that gives the opportunity to a user to enter inputs into the model and observe the outcome of the simulation. By this way the user gets artificial experiences during a very short time. In other word, WebSim with the system dynamics model in background, solves the problem distribution of factors that used to be a barrier for a user to get an overview of a complex system with a dynamic character.
Multi-player WebSims are brands-new and there are few multi-player simulations at the time being. Examples of some interesting multi player WebSims are Beer Game and Oil Fund which Beer Game will be discussed later in the next chapter.

2.3 WebSim development tools

Developing a WebSim like development of any other software interface requires some preparations and some tools. These preparations and tools are as follows:

- having a model that is constructed in Powersim Constructor;
- clarifying in advance the look of the WebSim interface and how the simulation should work;
- deciding the output and input variables;
- exporting the input and output variables by Powersim Exporter;
- developing the interface in Java (Borland JBuilder tool or other Java development tools like Visual Java++);

- embedding the designed WebSim in a html page and publishing it on a web server;
- starting Powersim Metro server in parallel to a web server on the same server machine;
- having a Java enabled browser like Internet explorer 4.0 or Netscape communicator 4.0 and browsing the website that embedded the WebSim;

It should be noted that the development of a model in Powersim Constructor and the design of the WebSim front-end in Borland JBuilder, are not necessarily done by the same person. But there is normally close co-operation between the model developer and the WebSim designer. These two tasks can be accomplished in parallel but it is normally better to have the model already built up in Powersim Constructor. The main reason for this is because of the complexity of WebSim development. It is easier to change or modify a model than modifying already developed interface and thereby modifying the code.

A good WebSim front-end design leads to better and new understandings of the model. The WebSim front-end can be considered as a gateway to the model.

In the following sections I will give brief presentations of the above mentioned tools.

2.3.1 Powersim Constructor

Powersim Constructor is the tool where the envisioned system is modeled and built up.

Powersim Constructor is based on system dynamics methodology. There exists other tools that can also develop models using the system dynamics technique. These tools are iThink, STELLA, Vensim, and DYNAMO.

WebSims can also be based on models constructed with other SD modeling tools (eg iThink/STELLA, Vensim and Dynamo), since a conversion tool exists that at least in some cases allow simple translation into Powersim format.

2.3.2 Powersim Metro Java API

Powersim Metro Java API (Application Programming Interface) consists of packages that are essential for the development of WebSims. These packages are both Java based and network capable. Metro contains packages that enable the

programmer to design WebSim front-ends with powerful objects like time-graphs, time-tables, gauges and many other objects and provides the possibility to the programmer to connect to the underlying model.

In this project I have used a pre-release version of an enhanced Powersim Metro Java API 1.1 that was developed for the purpose of programming the Beer Game WebSim.
In the next section I will present some main aspects of Powersim Exporter and explain its role in development of WebSim.

2.3.3 Powersim Exporter

When a programmer wants to begin the development of a WebSim and the model has already been constructed in Powersim Constructor, Powersim Exporter is the starting point.
Powersim Exporter provides additional information about the simulation and how the simulator (WebSim) should connect to the model. This information is needed by Powersim Metro Server and is stored in the file that contains Powersim model. Examples of this information are, how many slots a session has and which variables of the model are accessible to the WebSim.
A session is simply an instance of a simulation and slots are the entry points to the simulating model.
Each slot contains variables of the model that WebSim will have access to and has properties like label, type etc that can be set by the WebSim developer.
Another understanding of the slots is that they define the roles that players will take in a multi-player WebSim. For example in the popular multi-player simulator Beer Game, it is defined four slots for four different roles. These roles are retailer, wholesaler, distributor and factory.
As it is mentioned earlier each slot contains some variables that a player can have access to. The sort of access that the player can have to these variables is either as input or output. Each variable has some properties that can be set beforehand by the WebSim developer. It is in the properties that the sort of access for a variable is specified either as output (“Read” property) or both as input and output (“Read&Write” property).

It should be noted that these variables must be made public before the slot can use them at all. Powersim documentation is a good reference for learning how to do this.

Depending on what kind of multi-player simulator we are developing, the slots can either contain the same variables or different variables. If we are developing a multi-player simulator which has players with different roles then each slot will have its own specific set of variables that is different from the other slots. In this case the interface of the WebSim can be different from one player to another and it will be built according to the role that the player carries.

In some other cases of the multi-player simulator, all the players can have identical roles. For example they can all be managers or employees. In this case the slots will have similar set of variables. Accordingly the interface of the WebSim for all players can be the same. The multi-player simulator OrgSim is a good example of the last case.

2.3.4 Powersim Metro Server

According to the Powesim Metro documentation, Powersim Metro Server is a Windows Socket based server. This server allows client applications to access a running simulation across a network. Metro Server is capable of running both single- and multi-user simulations. The client application is called the WebSim front-end, while the connection between the front-end and the server is called a simulation stream.

The Powersim Metro Server must run alongside a web server on a main server. It means that both the Metro Server and the Web Server must be located on the same machine. This is caused by security restrictions in Java, that it doesn't allow an applet to connect to a server other than the server it came from.

The interaction between the Powersim Metro server, the web server and the WebSim fron-end can be described as follow:

When the user browses to a web page with an embedded WebSim, a connection between the web server and the client is established. Through this connection the web page with the embedded WebSim front-end (for example OrgSim) is sent from the server to the client. At this point, there is still no connection between the Metro Server and the client.

Once the WebSim front-end is downloaded to the client, the web browser executes it. During the initialization phase of the applet, the front-end sends a connection request to Metro Server. If the client has appropriate access permissions, a simulation stream is established. The WebSim front-end can now connect to an existing, or start a new a simulation. This is done by requesting the Metro Server to open a model (for example OrgModel.sim), and thereby open a session.

In the Metro Server we can actually see the opened session and its slots (see the figure below).

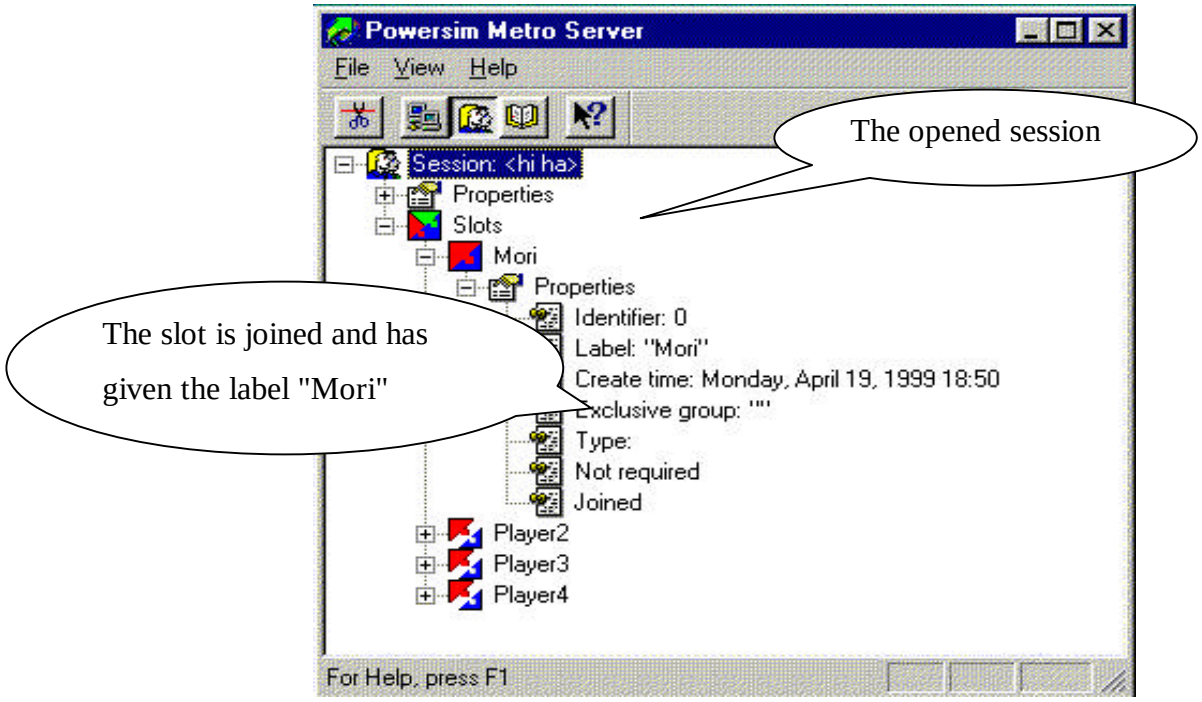

Figure 2.1 The Metro server

2.3.5 Summary Of WebSim Developments Tools

The figure below shows the tools that build the different parts used by a WebSim and how these parts are relatd to each other. Powersim Constructor builds the underlying model, Powersim Exporter exports the model variables and Borland JBuilder builds the WebSim front-end.

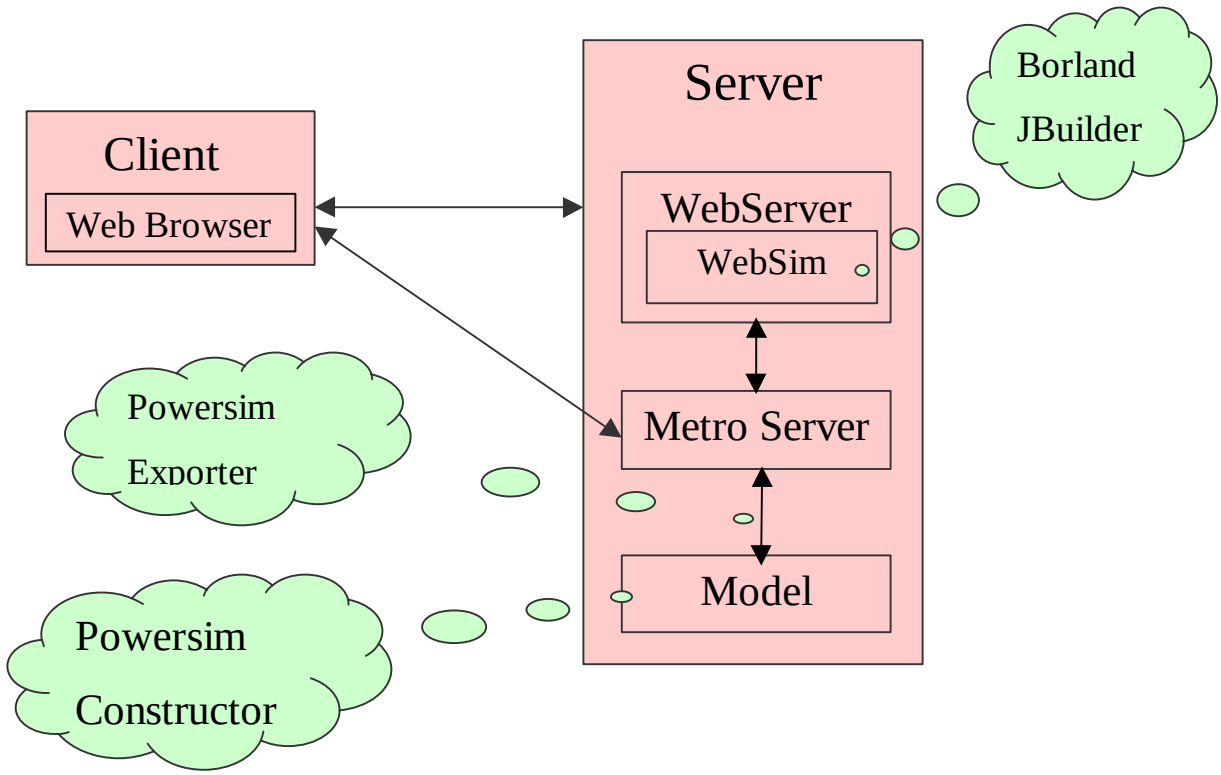

Figure 2.2 Overview of WebSim development tools.

The figure above also shows how WebSim, client's browser and the server interact with each other. This interaction is explained in the following:

When the user opens a web page with an embedded WebSim, a connection between the web server and the client is established. Through this connection the web page with the embedded WebSim front-end is sent from the server to the client. At this point, there is still no connection between the Metro Server and the client.

Once the WebSim front-end is downloaded to the client, the web browser executes it. During the initialization phase of the WebSim, the front-end sends a connection request to Metro Server. When the connection to the Metro Server is set up successfully, WebSim can either connect to an existing simulation (session), or start a new simulation. This is done by requesting the Metro Server to open a model (for example OrgModel.sim), and thereby open a session.

Chapter 3 PsWebsim Technology In Multi-Player WebSims

3.1 Introduction to Websim class and PsWebsim class

In the previous chapter we saw the necessary tools for developing a WebSim. All those tools were just providing the necessary preparations for the actual development of the WebSim front-end in Borland JBuilder.

Before proceeding with the process of building the multi-player simulator OrgSim in Borland JBuilder, I will go through two Java classes called "Websim class" and "PsWebsim class".

In the following when I refer to WebSim as an applet I will use the notation "WebSim" and when I refer to WebSim as a class I will use the notation "Websim class or PsWebsim class".

These two classes are used in the programming process for developing the WebSim.

Websim class extends (inherits from) PsWebsim class and is mainly used for development of single-user WebSims. This class simplifies the process of making a WebSim for a programmer and thereby the development process is shortened effectively.

PsWebsim class extends the standard applet class used to construct generic Java applets and is mainly used for development of multi-player WebSims.

PsWebsim class is rather a young technology that has been developed recently but it does not exist so much experience in using this class for developing multi-player simulators.

Therefore one of the goals set for this thesis, as it was mentioned earlier in the first chapter, is to test the ability of this technology.

But before going through PsWebsim class in detail, it is worthwhile to present the multi-player simulator "Beer Game" that was developed without implementing the PsWebsim technology.

3.2 The Multi-Player Simulator Beer Game

As a pre-project for this thesis, I did some research on the popular multi-player simulator ”Beer Game”. The purpose of the research was to see how the simulator was built up and how PsWebsim class could improve the simulator. The result of

the research was a PowerPoint document that gives some general guidelines in building WebSims and an overview of the Beer Game structure. Since Beer Game is a product made and owned by Powersim AS, I could not deepen my discussion with the details about this product. Therefore in the following I will just present an overview of the structure for this multi-player simulator without going in detail of how everything is initialized and developed.

Beer Game is a classic simulation developed at Massachusetts Institute of Technology´s Sloan School of Management in the 1960s [2]. The simulation model used to illustrate delays in supply chain management. Powersim AS made an interface for this simulation model using the Powersim Metro [3].

Beer Game extends directly the applet. The preliminary initializations for this simulator are done manually by the programmer. Two examples of these initializations are, connecting to the Metro Server and handling the System Messages.

What is special about the Beer Game is its interface. Almost every single object used in the interface is made by the programmer and is not found in Borland JBuilder development tool. These objects are initialized during the run time and not in the design time. This special feature of Beer Game has made the simulator very stable when it is run in different browsers. Some main parts of the structure for Beer Game are illustrated in the figure below. The yellow boxes show the classes made by the programmer

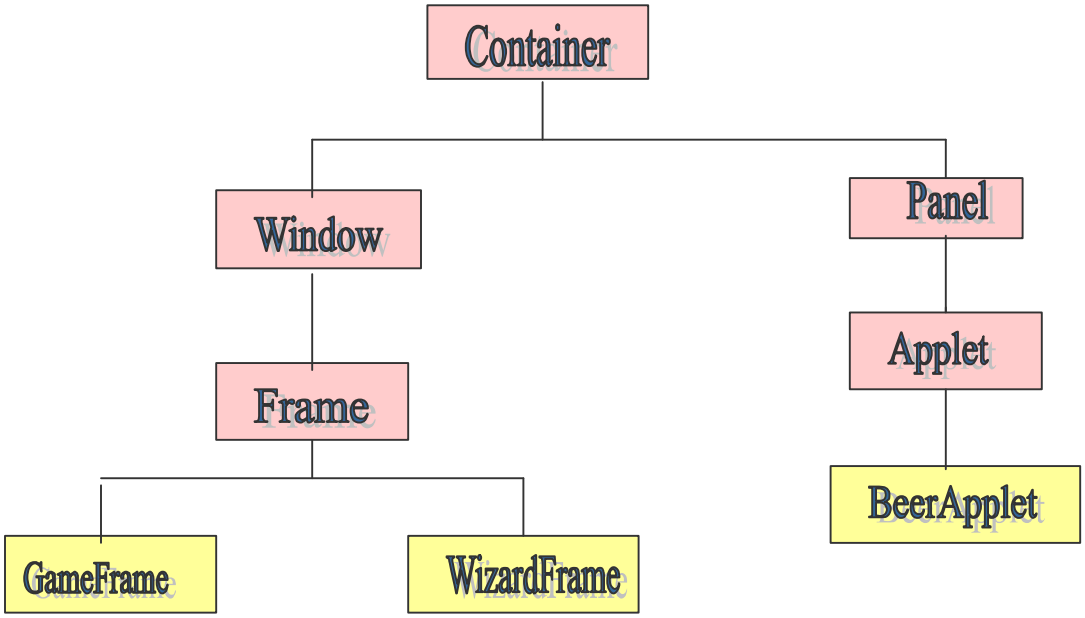

Figure 3.1 Parts of the structure for Beer Game

As we see from the above figure, BeerApplet extends directly the applet. It is from this class that Beer Game starts running. WizardFrame is a wizard widow which it collects some information from the user. GameFrame is the main play window where the user will play and can see the results of his play. The wizard window and the play window (GameFrame) are initialized one after the other and shown to the user. As we will see later OrgSim has almost a similar structure with the exception that it extends PsWebsim class and not the Applet class.
In the next section we will have a look on PsWebsim class and its main methods that can be used in developing a multi-player WebSim.

3.3 PsWebsim Class Structure

PsWebsim class is a file written in Java language by Powersim AS.
The main purpose of the PsWebsim class is to simplify the process of creating the multi-player WebSims by the programmer.
PsWebsim class extends the applet and implements the interface PsSystemMessageListener.
PsWebsim class can automatically handle Metro Server System Messages by displaying a dialog window showing the messages.
The WebSim developer can either use this default handling or overwrite OnPsSystemMessage to handle System Messages as desired. In OrgSim the default handling is used.
When PsWebsim is called by its subclass, it runs its init() method in order to carry out some initializations.
These initializations are as follow:

- It declares some variables which they will be used to save the applet and Websim parameters. These variables are:

  //Applet parametes to be cached
  protected int port;
  protected String host,path;
  protected boolean debug;

  //Information about the Websim parameters
  protected PsSessionManager session_manager;
  protected PsSession session;
  protected PsSlot model;

  After the declaration of variables the init() method continues with the initializations as follow:
- Gets the path to the sim model (the model which is created in Powersim Constructor);
- Gets the host (the IP address to the Metro Server which it will connect to);

- Gets the port that it will communicate over;
- Creates a debug window to show the Metro Server System Messages. It should be noted here that this window will only be shown when the parameter debug is set to true inside the applet tags for the html page where the simulator came from.
- The last initialization here is to connect to the Metro server by calling the following method:

CreateSessionManager();

When a session manager is created successfully, it means that a connection to the Metro Server is set up and PsWebsim goes back to its subclass to where it had been called from (As we will see later it is from init() method for subclass which the super class PsWebsim is called).

PsWebsim class provides some useful methods that the programmer can call them in the process of developing the WebSim. These methods are as follow:

CreateSession();

CreateSession(String password);

ConnectSession();

Connect Session(String password);

Join (long id);

Join();

destroy();

getModel();

getSession();

getSessionManager();

All these methods will be tried later in the development of OrgSim. To see the code details of these methods refer to OrgSim program .

Chapter 4 The Process Of OrgSim Development

4.1 The Design Requirements For OrgSim Front-End

Before commencing with the programming and developing of OrgSim, we have to specify the look of OrgSim front-end and how the user will interact with this interface.

OrgSim is an applet and it will be embedded in a web page. When a user browses this page the applet will be downloaded to his machine. The downloading process usually takes time. Therefore we may need to put some introductory information about OrgSim in the web page that embeds OrgSim while the user is waiting for downloading process to get finished.

Several simulation sessions can be run for the underlying model "OrgModel".

In each session of simulation several players can play against each other. In OrgSim I have chosen to limit the number of players in each session to four. Since we have four players in each session, we may provide the names of the competing players to each participating player in the session.

As it is mentioned several simulation sessions can be run for OrgModel. Then a user can either create a new session of simulation or join an existing session. Since several sessions can be created by different users, we need to separate between these sessions. This can be done by letting the user name the new session that he is creating. When the sessions are named, it will be possible for the other users to join a specific running session.

The above specifications can be designed by creating a wizard window that collects the necessary information from the user.

The wizard window has three screens. In the first screen of the wizard window the user has to choose either to create a new game or to join an existing game. Depending on his choice either the second or the third screen is shown to him. For example when he chooses to create a new game and clicks on the next button, the second screen appears for him. In this screen he may enter his name and a name for the session that he wants to create. When he clicks on the next button again, the wizard window closes itself and a new window will be opened (this new window is the main window to play and is explained below in the next paragraph). When the user chooses to join an existing game in the first screen of

the wizard window, then the third screen is shown to him. In this screen he may provide his name and choose a session from the existing running sessions that are shown in a list box. When he again clicks on the next button, the wizard window is closed and another window (this is the main window to play and is explained in the next paragraph) is shown to him.

OrgModel has many output and input variables. A user may enter his decisions to some input variables and click a button that starts the simulation. The results of the simulation can be shown by the output variables. Java API objects like time graph and time table can be used to present these output variables.

It is desired that the input and output variables will be placed in one window. To do so the tabsetPanel object for Borland JBuilder tool is used and placed in one window. TabsetPanel object consists of tab panels. The desired input and output variables can be placed on these tab panels. By this way we have managed to show all the input and output variables in one window without crowding the window with different variables.

The result of the above implementation is a front-end that has two windows. One is wizard window and the other one is the main window for playing. The wizard window collects the needed information from the player and when it is done it closes itself and opens the main window where the player can play and see the results of the simulation.

In the next section we will go through some preliminary preparations for OrgSim development.

4.2 Some Development's Preparations

To start the actual development of OrgSim, we need to carry out some preparations first here.

These preparations are setting up a suitable folder structure for OrgSim, enhancing the OrgModel, and exporting the model variables by Powersim Exporter.

In the following section we begin with the first preparation that is setting up a structure for the folder that will contain all the files for OrgSim.

4.2.1 Folder Structure

Because of the file naming conventions used by the Java language, having a good folder structure for placing the source files is essential for the programmer of the WebSim. Setting up a standard and following it up during the WebSim development, makes it easy for developer to locate files later and update them. Many companies use a basis folder for software development that is called Dvl\ or Dev\ or whatever it is convenient for them.

In our project we follow the same standard and call the basis folder WebsimDvl\. When we update the project we save it as WebsimDvl1 and WebsimDvl2 etc. By this way we have some backups of the project. The last version of the project was saved under "WebsimDvl10\" and since it was the last version that I produced, I changed the name of the folder to WebsimDvl\.

In addition to this basis folder we need to create some subfolders that conform with JBuilder's path definitions for source (i.e.java) and output (i.e. class) files. These subfolders are "\WebsimDvl\Java\src" and "\WebsimDvl\Java\lib " respectively. Pictures are placed in the subfolder "\WebsimDvl\Java \Images".

The next preparation is to choose an appropriate name for the package that will hold all the classes and other packages of OrgSim. Here we follow the current convention for naming the packages. According to this convention the first level of the hierarchy specifies the globally unique name of the company that develops the Java package. For this we add the subfolders "\bt\orgsim" under "\WebsimDvl\Java\src" and "\WebsimDvl\Java\lib "

The figure below shows the folder structure for Orgsim.

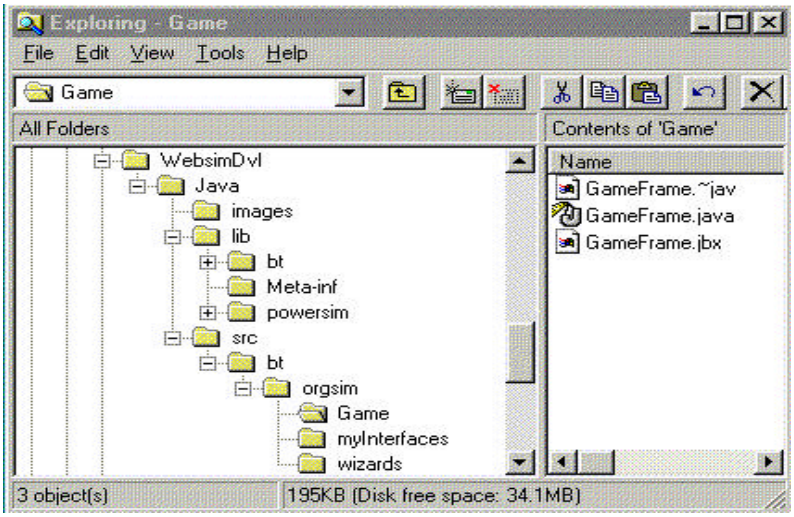

Figure 4.1 The Structure of the folder.

4.2.2 Enhancing OrgModel

The model that I will be using for OrgSim is constructed in Powersim Constructor. This model is about how knowledge management affects organizational performance.

The key message that the model tries to promote is where you invest in knowledge depends on environmental conditions. In a stable environment the focus is on efficiency and in a rapidly changing environment the focus is on effectiveness and flexibility.

From now on I will refer to this model as OrgModel.sim or just OrgModel. OrgModel is an arrayed model that means in each simulation several players (organizations) can play against each other.

Before I could use OrgModel in developing OrgSim, some necessary modifications had to be done. The first one was specifying the number of players in the game to be 4.

The second modification on the model was to replicate the situation where the player and the model can take over the simulation from each other when it was necessary. This situation is modeled as shown in the figure below.

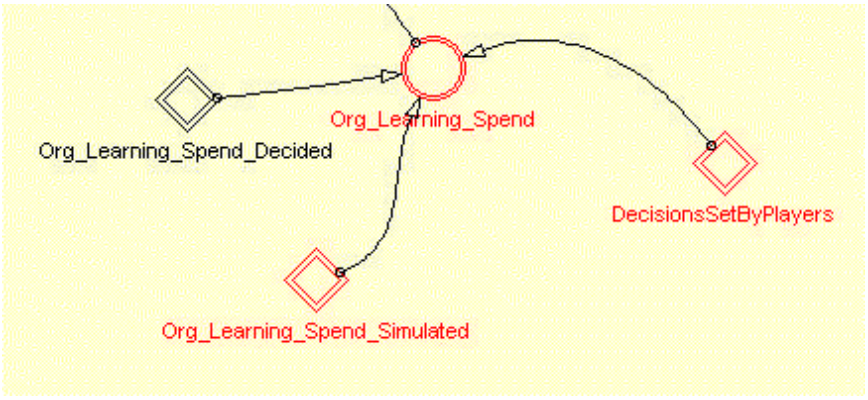

Figure 4.2 Enhancing the model

The red colored variables are those that have been modified. The two constants "DecisionsSetByPlayers" and "Org_Leaning_Spend_Simulated" are added to the model. The constant "DecisionsSetByPlayers" takes the value 0 or 1. Depending on the value of this constant, the variable "Org_Learning_Spend" will choose either to use the player's decision or the model decision which in this case is the constant "Org_Learning_Spend_Simulated".

The constant Org_Learning_Spend_Decided is an input variable in OrgSim and is initialized to 50 (this variable has the label Knowlegde Investment in OrgSim). The player enters his decision into this constant and if "DecisionsSetByPlayers" is set to one then the variable "Org_Learning_Spend" will choose the entered player's decision for the simulation otherwise it will use "Org_Leaning_Spend_Simulated" value for the simulation. The constant "Org_Leaning_Spend_Simulated" is initialized to 50.
After modifying the model, we setup a connection between OrgSim and the model variables using Powersim Exporter.

4.2.3 Exporting The Model Variables

As it is mentioned earlier in the previous sections OrgModel is arrayed and simulated for four players. This means that in each simulation four players can compete with each other. Therefore in Powersim Exporter we need to define four slots which each will represent a player. As the figure below shows four slots are defined each with a default label “player1” or “player2” etc.

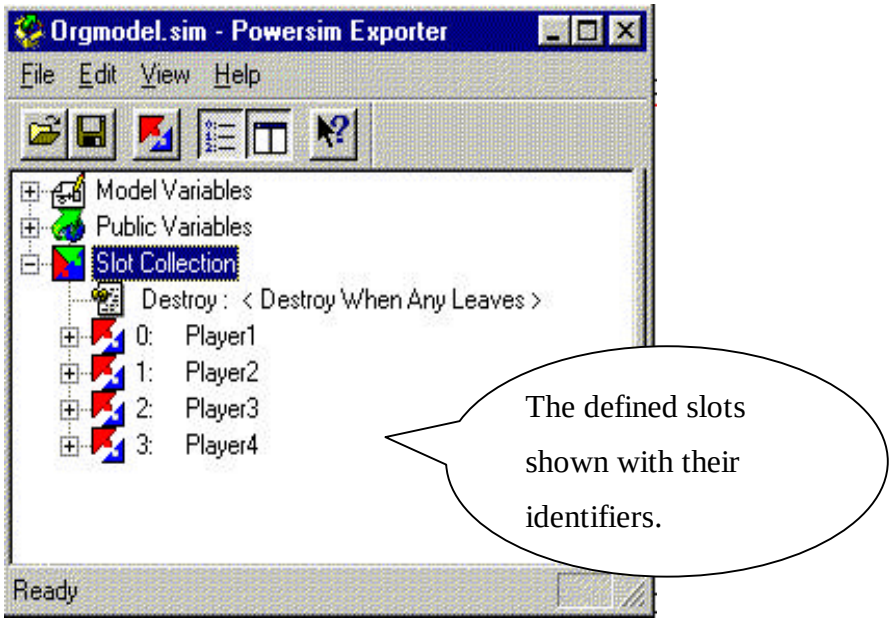

Figure 4.3 The slots in the Exporter

But before we can put any variable in these slots we have to make the model variables public. This is simply a dragging and dropping action. After making the variables public we can now decide which variable goes to which slot. Since all

the players in OrgSim have the same role therefore all the slots will contain the same set of variables.

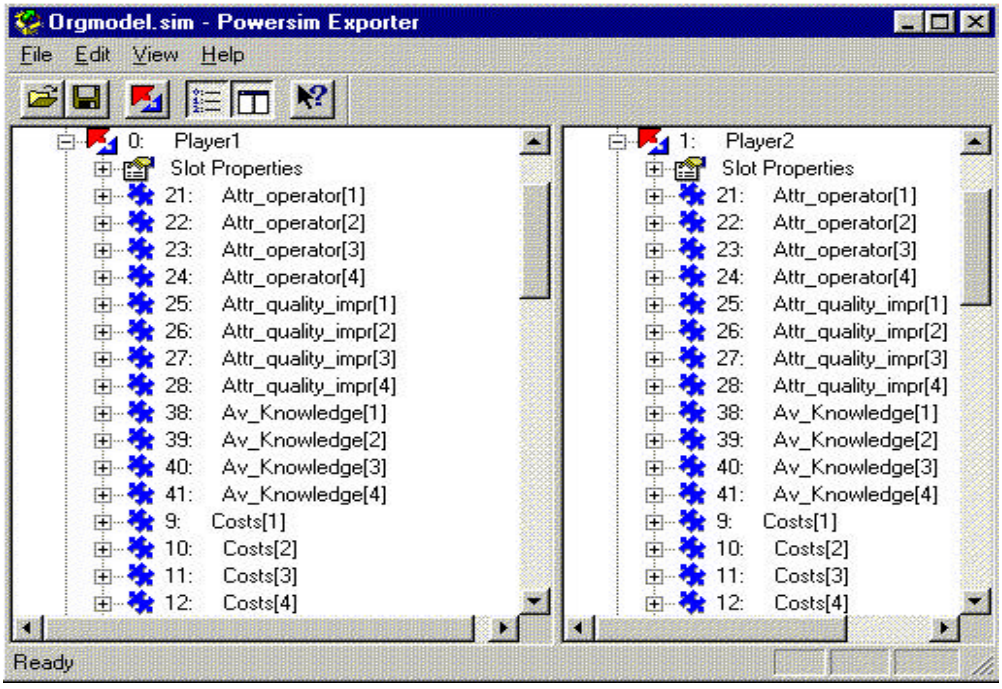

Figure 4.4 The slots' variables

There is an important property here that should be set right. This property belongs to “Slot Collection” and indicates what will happen when a player leaves the simulation. This property has three values that one of them will be set for “Slot Collection”. The required functionality in OrgSim is when one player leaves the game the slot belonging to this player should be emptied without affecting the ongoing simulation. When the last player leaves the game, the session should be cleaned and removed from the Session Manager it belongs to (some codings are required here that will be discussed in chapter 4 section 4.3.3.3).
The right property for “Slot Collection” in this case is “ Destroy when all leaves”.

4.3 The Class Structure Of OrgSim

The multi-player simulator OrgSim has a class structure that is more or less inspired by Beer Game class structure. Parts of the Beer Game class structure was shown in chapter 3 in section 3.2.
Figure 4.5 below represents the class structure of OrgSim as a whole. In this figure we can see how the different classes in OrgSim are related to each other.

It should be reminded that when I use the notation "OrgSim", I am referring to the multi-player simulator as a whole and when I use the notation "Orgsim", I am referring to the Orgsim class.

In the figure below the yellow boxes contain the classes that I created manually and the rest (except PsWebsim class that is made by Powersim) are Borland JBuilder classes which have already been built.

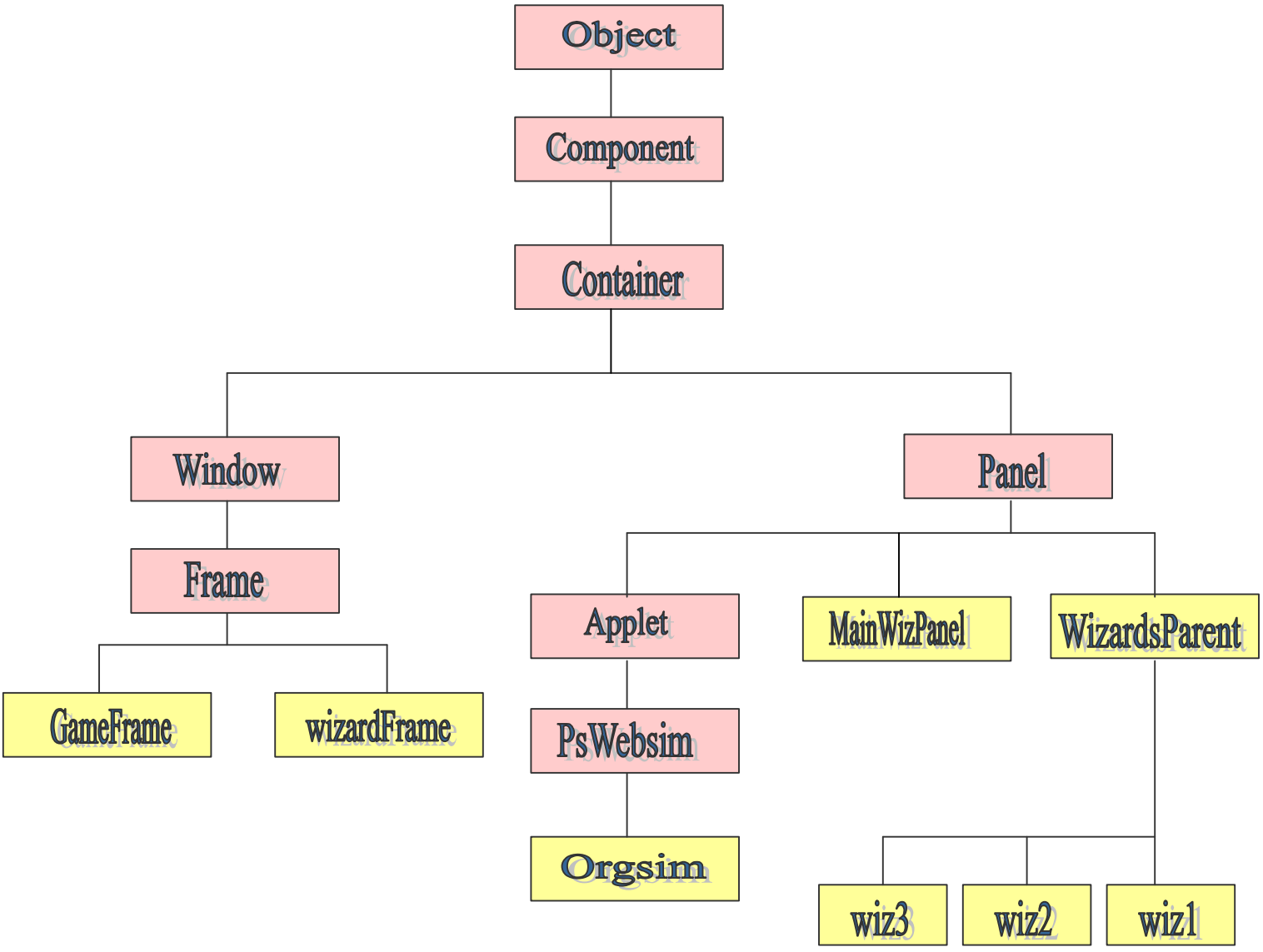

Figure 4.5 The Class Structure of multi-player simulator OrgSim.

The figure above only shows the classes found in OrgSim with their super classes or sub classes but does not say how they are used or being called or initialized. In the following sections issues like, where these classes are placed, which class initializes which, when they are called and so on, are described in detail.

4.3.1 Orgsim Class

OrgSim simulator begins its running from Orgsim class. This class extends PsWebsim class (see figure 4.5).

The first notable thing that can be seen here is the start of the class. It begins with the following line:

package bt.orgsim;

This line of code means that the class Orgsim will be contained in this package. This code comes automatically up if it has been specified during the process of creating the applet in JBuilder. Before we start writing any code, we have to import the Powersim Metro Java API classes. These classes are as follow:

```
import powersim.Metro.lang.*; //Basic classes to create WebSims
import powersim.Metro.event.*; // Handles events generated by simulators
import powersim.Metro.simul.*; //Highest level approach to building simulators
```

The super class PsWebsim does the first initializations for Orgsim class. To carry out these initializations, Orgism class must call this super class from its init() method as shown below.

```
public void init() {
  super.init();
}
```

The first vital initialization for Orgsim class (and thereby for OrgSim simulator) is to set up a connection between the local machine and the Metro Server (this connection is set up by PsWebsim class during the preliminary initializations).

From Orgsim class two other important classes are initialized. In Java Language, it said that Orgsim class is an object that contains two other objects. The figure below shows the objects found in Orgsim object (label, textboxes and etc are also called objects).

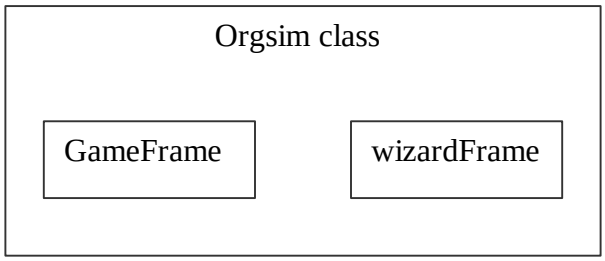

Figure 4.6 The main objects in Orgsim class

When Orgsim class has managed to establish a connection to the Metro Server, it begins its own initializations.

The first one is the initialization of the wizard window.

This initialization is done in the jbInit() method by calling the constructor for wizFrame class as shown below:

userWizards= new wizardFrame(this);

Here it is presumed that the class "wizardFrame" is already built and we just make an instance of this class (object). Besides we have to import this class to Orgsim class so Orgsim class can find it. This is done in the top of Orgsim class by the following code:

import bt.orgsim.wizards.;*

What happens inside this class when it is initialized, will be explained in the following section. It is enough to know here that this class is responsible for showing the wizard window.

The purpose of the wizard is to collect some information from the player before creating any game. When the information is collected, the wizard calls the method *mainGameStarter()* for Orgsim class. It is this method that initializes the GameFrame class (the game window where the player actually can play and see the results of his play by time graphs and time table).

There are two constructors for GameFrame class in this method. Depending on the information collected from the wizards one of these constructors will be executed. One constructor is for creating a new game and the other for joining an existing game that is in progress.

If the player has chosen to create a new game the following constructor is called:

GameFrame mainGameFrame =new GameFrame(this,playerName,teamName);

As we see from the above constructor the name of the player and the team along with Orgsim object (this) is delivered to the GameFrame class.

If the player has chosen to join an existing game then the following constructor is called:

GameFrame mainGameFrame =new

GameFrame(this,playerName,teamName,session);

This constructor delivers Orgsim object (this), the name of the player, the name of the team that he wants to join, and the session belonging to this team.

But before calling the second constructor for joining an ongoing game two things happen first. The first one is to find the session that the player wishes to join and the second one is to check if this session is full or not. These tasks are done by the following methods:

getExistingSession();

checkSession();

The details of these methods are found in the appendix of this report.

By executing one of the above mentioned constructors for GameFrame class, the program goes out of Orgsim class and continues its work in the GameFrame class. The diagram below shows how Orgsim class executes when the applet (OrgSim simulator) is downloaded.

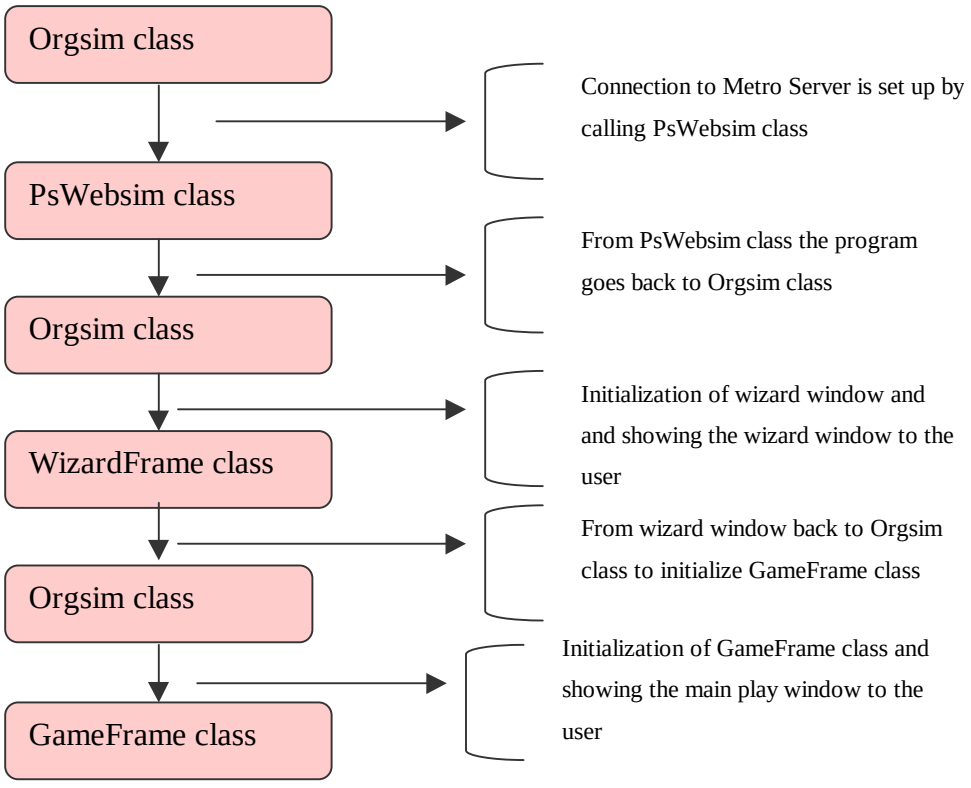

Figure 4.7 the process of initializations

In the next section I will explain the wizardFrame class and the objects contained in this class in detail.

4.3.2 wizardFrame Class

The wizardFrame class extends the Frame class and implements an interface called ExistWizard.
wizardFrame class is placed under the following package:
package bt.orgsim.wizards;

This interface ExitWizard has only one method which wizardFrame has implemented it in its class. This method is called *btExitClicked()*. I will come back to this method in the next sections to show which object calls this method and from where.
The main purpose of wizardFrame class is to show all the wizard objects in one window. But the question is here how the wizard objects are placed on the wizardFrame. Actually wizardFrame contains only one object called mainWizpanel. It is mainWizpanel that contains the most of the wizard objects. In the figure below we see how the wizard objects are placed in one window (wizardFrame).

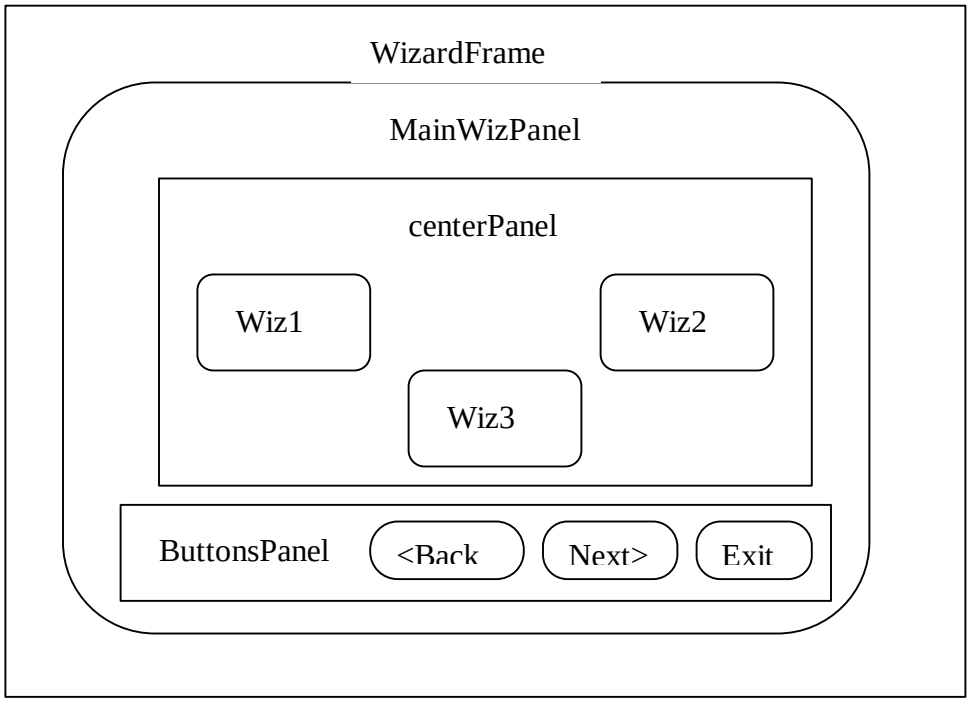

Figure 4.8 The wizards structure as a whole

In the jbInit() for wizardFrame class the constructor for MainWizPanel is called as follows:
mainWizpanel=new MainWizPanel(OrgWebsim);

The class Orgsim was delivered to the wizardFrame class as an object (see the constructor for wizardFrame class in Orgsim class where it was called for the first time). WizardFrame saves Orgsim object in the variable OrgWebsim. The constructor for MainWizPanel takes this object as an argument and delivers it to the mainWizPanel class. This object will be used there to call mainGameStarter() method for Orgsim class.
When the constructor "MainWizPanel(OrgWebsim)" is called the program goes to the MainWizPanel class to execute and initialize this class.
In the next section we will have a look on the details of this class (MainWizPanel).

4.3.2.1 MainWizPanel Class

This class extends the panel and is located under the following package:
package bt.orgsim.wizards;

As it is seen in the figure 4.9 this class contains two main panels that called centerPanel and ButtonsPanel. The centerPanel contains three objects that are wiz1, wiz2 and wiz3.
These objects are instantiated in the jbInit() method as follow:
wiz1User= new wiz1();
wiz2User= new wiz2();
wiz3User=new wiz3();

After instantiation they are placed on the centerPanel with wiz1 shown as default:
centerPanel.add(wiz2User,"wiz2");
centerPanel.add(wiz3User,"wiz3");
centerPanel.add(wiz1User, "wiz1");
cardLayout1.show(centerPanel,"wiz1");

The layout for centerPanel is *CardLayout* and because of this layout it has become possible to show the object wizards placed on the centerPanel one at a time.
The ButtonsPanel contains three buttons that are "Next", "Back" and "Exit" button.
The next and back buttons page forth and back between these wizard objects (wiz1, wiz2 and wiz3). After gathering all needed information from the player, the method mainGameStarter for Orgsim class is called from within the Next button. To see how Next and Back buttons work, refer to the code for these buttons in MainWizPanel class in OrgSim program.
Before discussing the wizard objects wiz1, wiz2 and wiz3, let's have a look on their super class WizardsParent.

4.3.2.2 WizardsParent Class

WizardsParent class extends the panel and placed under the following package:
package bt.orgsim.wizards;

This class is the super class for the wizard objects wiz1, wiz2 and wiz3. For the time being this class has only two methods getNextPanel() and getPrevPanel(). The reason for having a common super class for the wizard objects is because of their similar functionalities. It is easier to write methods for one class and inherit this class instead of writing the same methods several times in different classes. Besides for further development in the future, it will be easier for the programmer to put more functionalities for these three wizard objects.

4.3.2.3 Wiz1 Class

This class extends WizardsParent and is placed under the following package:
package bt.orgsim.wizards;

The figure below shows the interface for this wizard object.

Figure 4.9. The interface of wiz1.

This wizard object is the first one that appears for the player when running OrgSim simulator. As the figure implies, the player chooses whether to create a new game or to join an existing game.

This class has only one method that overrides its super class method. This method is getNextPanel(). This method simply returns the name of the next wizard object that will be shown to the player that is either wiz2 or wiz3. This method is called from "Next" button.

4.3.2.4 Wiz2 Class

This class as wiz1 extends WizardsParent and is placed under the following package:

package bt.orgsim.wizards;

The figure below shows the interface for this wizard object.

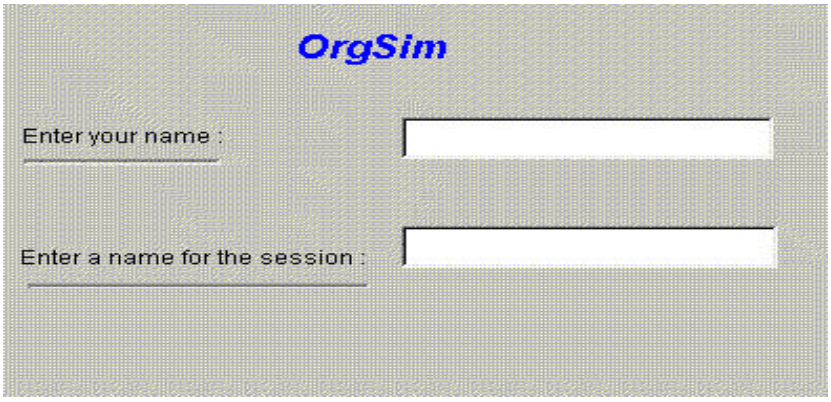

Figure 4.10. The interface of wiz2.

As the figure above shows the player can write his name or any other name that he wishes to use under simulation. He can also write a name for the session that he is going to create.

This class has only three methods that are as follow:

- getPrevPanel() that overrides the method for its super class WizardsParent;
- getName() that returns the name of player;
- GetTeamName that returns the name of the session;

The above methods are called from either "Next" button or "Back" button.

The next and last wizard object that we are going to see in the next section is wiz3.

4.3.2.5 Wiz3 Class

The last wizard object is wiz3. Like the other wizard objects this wizard extends WizardsParent class and is placed under the following packag;

package bt.orgsim.wizards;

The figure below shows the interface for this wizard object.

Figure 4.11. The interface of wiz3.

As we can see from the figure above, the player writes his name and chooses an existing game from the drop down list box. The label with cyan color appears when there is no game to join. This label will contain a message that says " At the time being there is no game to join.".

Wiz3 has three methods that are:

- getPrevPanel() returns the previous wizard object;
- getName() returns the name of the player;
- fillTeamNames() fills the listbox with the existing games (refer to OrgSim program to see how this is done);
- getTeamName() returns the selected game name;

All these above methods are called from either "Next" button or "Back" button.

4.3.2.6 Summary Of The Wizard Windows

As it is mentioned in wizardFrame class all these wizard objects are shown in one window.

The figure below shows how wizard1 appears in a window along with the buttons in Netscape Communicator.

Figure 4.12. The interface of wiz1 along with the buttons

4.3.3 The GameFrame class

We have been through Orgsim class and wizardFrame class in details in the previous sections. In this section I will explain the main issues of GameFrame class, issues like how the interface is built up, how the interface and the simulation communicate with each other and how the players relate to each other. In the figure below a part of the interface for this class is shown.

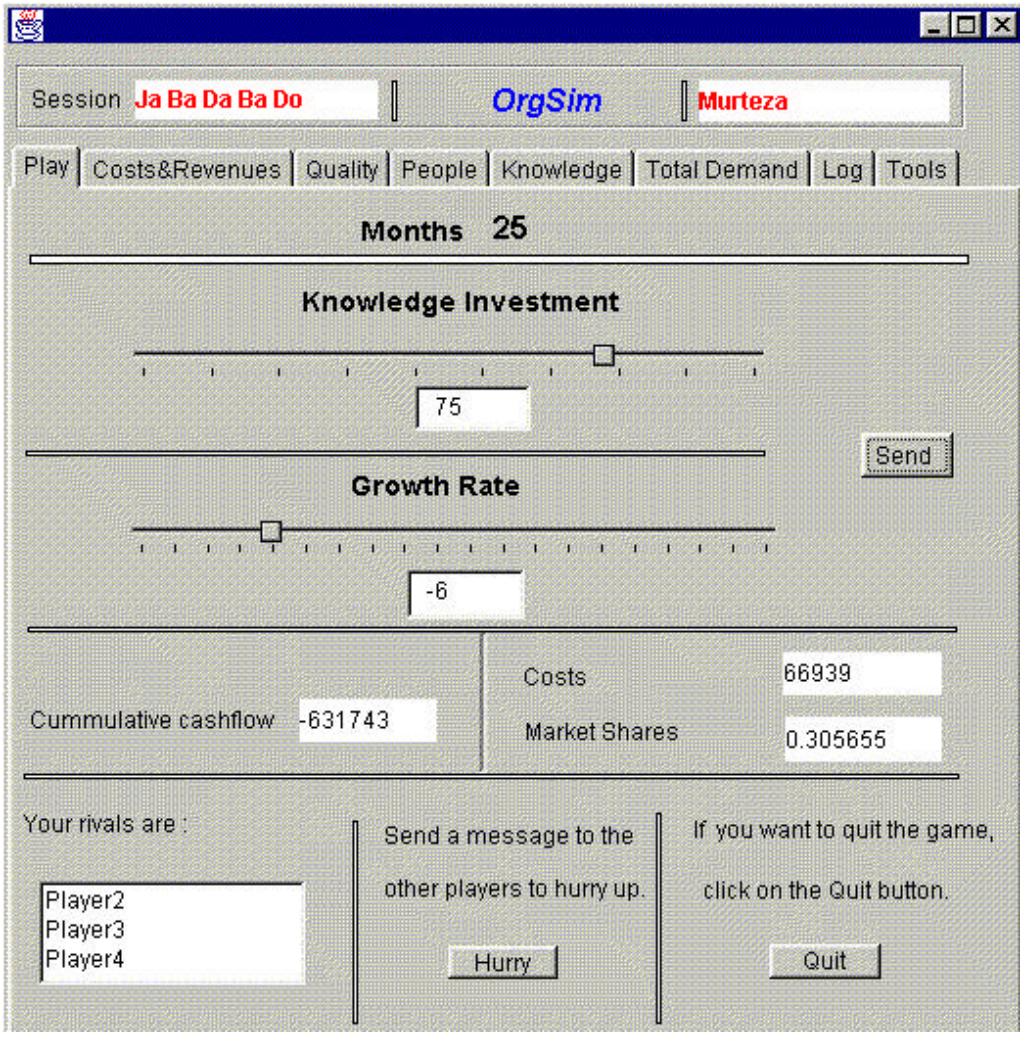

Figure 4.13 The play window for Orgism

In the first following section we will have a look on the interface details and see how it is built up.

4.3.3.1 The Front-End Of GameFrame Class

GameFrame class extends the Frame and implements several Powersim Metro interfaces. These interfaces are PsAdvanceListener, PsLeaveListener, PsMessageListener and PsJoinListener. These interfaces will be explained later in the following sections.

The GameFrame class is placed under the following package as it is seen on the top of this class:

package bt.orgsim.Game;

The Powersim Metro classes that are imported here are as follows:

```
import powersim.Metro.lang.*;            //Basic classes to create WebSims
import powersim.Metro.event.*;           // Handles events generated by simulators
import powersim.Metro.simul.*;           //Highest level approach to building
simulators
import powersim.Look.beans.PsTimeSeries;  // Necessary to create tables&graphs
import powersim.Look.beans.*;
import powersim.Metro.lang.*;
import powersim.Look.basic.*;
import powersim.Look.enum.PsLineStyle;
```

To integrate the input and output variables all in one window, the object TabSetPanel has been used. The figure below shows the general structure of the interface.

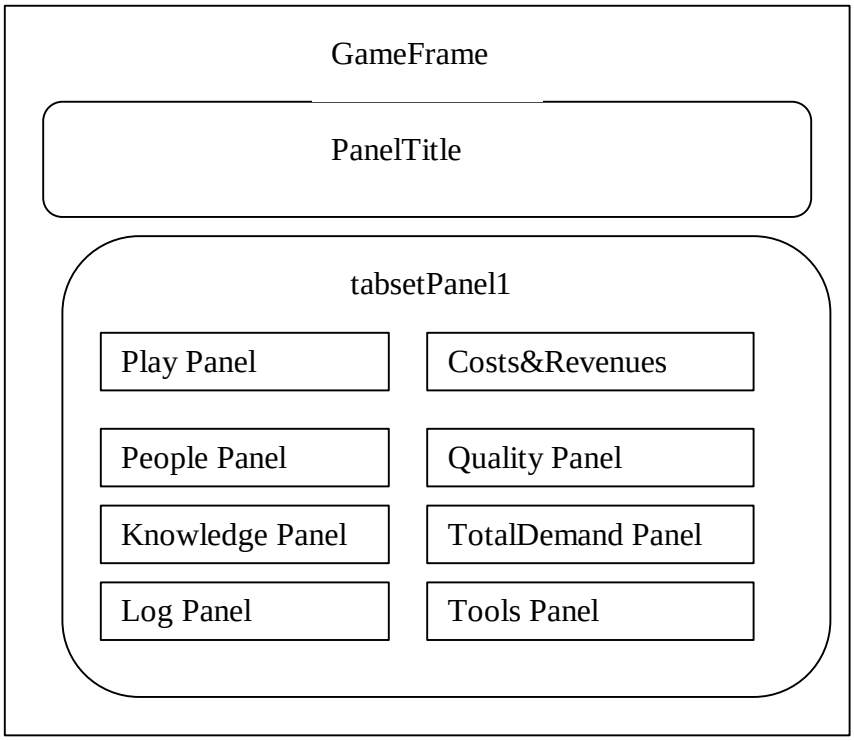

Figure 4.14 Overview of the interface structure.

Most of the above structure is initialized in jbInit() method and the rest of the initializations are done in GameFrame constructors.

Two of the most important objects in this interface which show the output results are the Powersim Metro objects "time graph" and "time table". The time graph object are found in the most of the tabsetPanels except for Play panel and Log panel. The time table object is found just in Log panel.
To be able to use these Powersim Metro objects their relevant class should be imported to GameFrame class. This class as we saw earlier in the beginning of this section was:
import powersim.Look.beans.PsTimeSeries; // Necessary to create tables&graphs

The first time initializations of these objects occur in the jbInit() method. In the following, time graph initialization for "Costs&Revenues" is shown as an example. To see the initializations for the other output variables (which have similar initialization) refer to the jbInit() method for GameFrame class in OrgSim program.

GraphCostRevenue.getTimeGraph().getLegend().setVisible(true);
GraphCostRevenue.getTimeGraph().getYAxis().getLabel().getTextFormat().setText("Value");
GraphCostRevenue.getTimeGraph().getXAxis().getLabel().getTextFormat().setText("Months");
GraphCostRevenue.getTitle().setVisible(false);
panel2.add(GraphCostRevenue,BorderLayout.CENTER);

In the figure below the "Costs&Revenues" panel is shown with a time graph placed in this panel to show the results for Costs and Revenues.

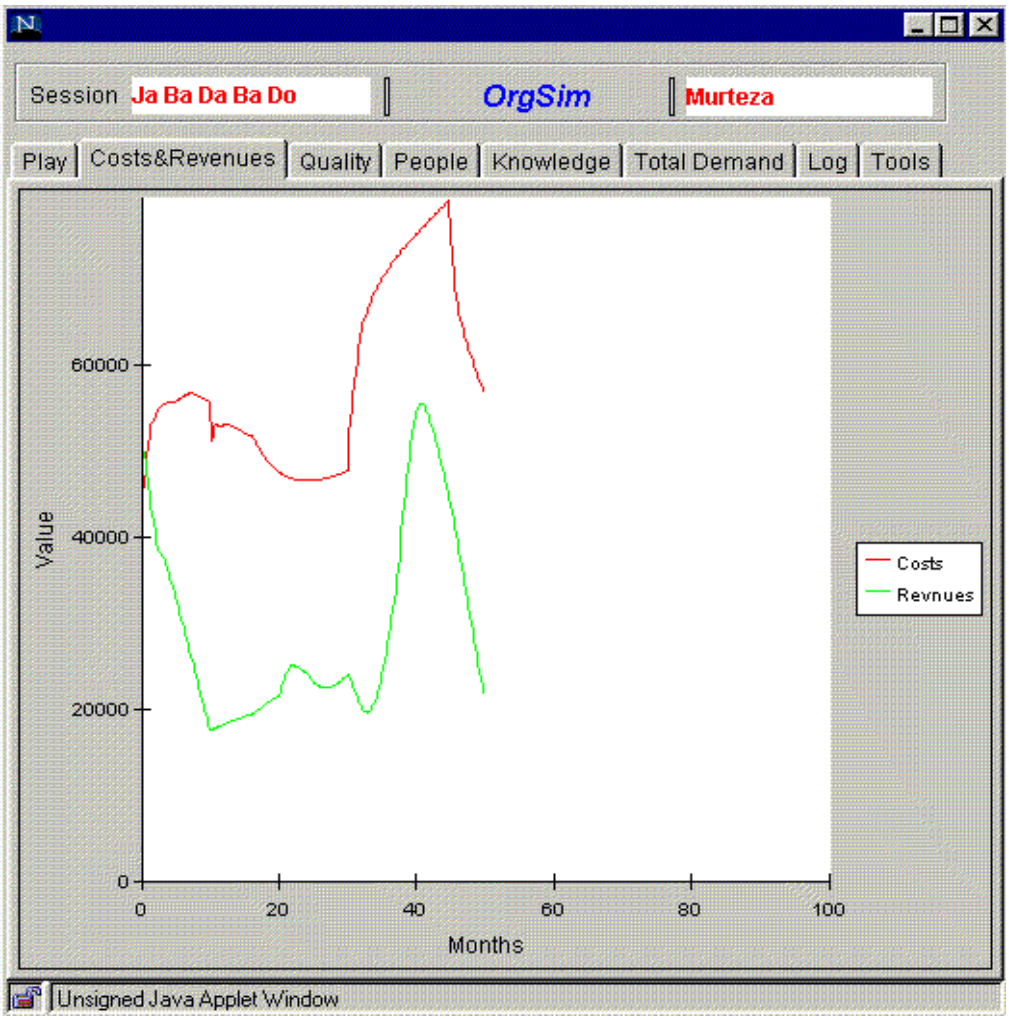

Figure 4.15 Time graph for costs and revenues.

How the graphs are connected to the model and thereby to the simulation is explained in the following sections. Let us see first how the constructors for GameFrame class work and what they do.

4.3.3.2 The GameFrame Class Constructors

GameFrame class has three constructors and two of them are the engine of this class, since calling one of them from Orgsim class, initializes the GameFrame class and runs it. As we have seen earlier in section 4.2.1 (Orgsim class), one constructor is designed for creating a new game and the other one is designed for joining an existing running game. Let's first have a look on the first one and see how it works.

Constructor For Creating A New Game

When the player has decided to create a new game, this constructor is called from the method mainGameStarter in Orgsim class. This constructor is as follows:

public GameFrame(Orgsim OrgWebsim,String playerName,String teamName)

The first thing that this constructor does is running another GameFrame class constructor to do some preliminary initializations like running the jbInit() method. This constructor is called *GameFrame()* and is simply called by the following code:

this();

After running the constructor GameFrame(), the constructor for creating a new game continues its executing by saving the received arguments from the Orgism class into some local variables.

this.OrgWebsim=OrgWebsim;

this.playerName= playerName;

this.teamName=teamName;

As we see from the above code, Orgsim class is saved as an object into the variable OrgWebsim. One question is raised here and it is that Why we are using Orgsim class in the GameFrame class. The answer is simple. We want to use the PsWebsim class methods here to facilitate the creation of a new game and since Orgsim class is a subclass of PsWebsim class so we access this super class via its subclass Orgsim.

Before we can create a session we need to get the SessionManager object that was already created by PsWebsim class during the first initializations of Orgsim class. This is done by using the PsWebsim method getSessionManager() as follows:

this.sm=OrgWebsim.getSessionManager();

After getting hold of the SessionManager, we use its method createSession() to create a new session as follows:

session=sm.createSession(ModelName);

The label of the session is set to teamName that the user had entered in the wizard window.

session.setLabel(teamName);

After creating the session, its existing slots are enumerated. Then it tries to find an empty slot and when it finds an empty slot, it saves it in the variable model that is of type PsSlot.

```
slots=session.enumSlots();          //the slots are being enumerated
boolean joining;
PsSlot model;
for ( int i=0; (i< slots.length) && (model==null);model=slots[i];
model= slots[i];
}
```

The empty found slot is joined by the following code:

model.join(playerName,this);

As the above code shows the slot label is set to the player name

In the above I have not reproduced all the details of the codes. Refer to OrgSim program to see the details for this constructor.

I have mentioned earlier that jbInit() method does just some of the initializations and not everything. The rest of the initializations are done via three methods that exist in this GameFram constructor. These methods in order of execution are as follow:

ParameterInitializer();

PowersimGUIinit();

tabsetPanel2Initializer();

ParameterInitializer()

The objects found in the Play panel (see figure 4.14) like labels, text boxes, sliders and etc are connected to the variables of the model in this method. All the objects are initialized and show the start value of the variables. The following is just an example of what this method does.

txtKnowInvest.setText(""+(int)model.getPar(1).getValue());

Refer to OrgSim program to see the details for this method.

PowersimGUIinit()

We have seen earlier how the graphs were defined and set in the jbInit() method. The next step is to connect these graphs to some variables. This method does this work for us.

Among the several methods that found in this method, one is worthwhile to mention. This method called *rememberSeries()* and keeps a record of the history of all values taken by a variable in the model. The following is an example of connecting the graph object "GraphCostRevenues" to a model variable that has 9 as its identifier.

model.getPar(9).rememberSeries(1,true);

parCosts = model.getPar(9);

GraphCostRevenue.addParameter(parCosts);

GraphCostRevenue.getTimeGraph().getCurveFormat(parCosts).getFillFormat().setForegroundColor(new PsColorFormat(255,255,255));

Refer to OrgSim program to see the details of this method.

Constructor For Joining An Existing Game

The second important constructor for GameFrame class is designed for joining an existing running game. When the player decides to join an ongoing game, this constructor is run from the mainGameStarter method in Orgsim class.

GameFrame(Orgsim OrgWebsim,String playerName,String teamName,PsSession session)

As the other constructor (the constructor for creating a new game), the first thing that this constructor does is to run the constructor GameFrame() by the following line:

this();

After running the constructor GameFrame(), the constructor for joining an existing game continues its executing by saving the received arguments from the Orgism class into some local variables.

this.OrgWebsim=OrgWebsim;
this.playerName= playerName;
this.teamName=teamName;
this.session=session;

One of the received arguments is a session. This session is the one that the user has chosen in the wizard window and wants to join. Recalling from Orgsim class, we saw that before calling the constructor for joining an existing game, two things happened first. The first thing was to find the session that the user has chosen by method getExistingSession(). The second thing was to check of this session had empty slot to join by the method checkSession().
When these two methods were done, the found session was delivered to the GameFrame class by calling its constructor for joining an existing game in Orgsim class.

The slots for the received session are enumerated and an empty slot is found and joined.

```
slots=session.enumEmptySlots();
for(int i=0 ;(i<slots.length)&&(model==null);i++){
model=slots[i];
}
```

Again the methods *ParameterInitializer(), PowersimGUIinit() and tabsetPanel2Initializer()*.are called to carry out the rest of the initializations for GameFrame class (see above in the previous sections).
Refer to OrgSim program for more details about this constructor.

Alternative Ways For Creating Or Joining A Session

As we saw in chapter 3 (section 3.2), PsWebsim class provides methods for creating and joining a session.
Some of these methods were tried inside the above constructors (the constructor for creating a new game and the constructor for joining an existing game) but not fully investigated due to the lack of time.
Methods like *createSession()* that creates a new session and *Join()* that joins an empty slot, were experimented with in the constructor for creating a new game.

Using these standard methods simplify development but pose a number of restrictions on the WebSim.
For joining an existing session, PsWebsim class has two methods for it. These methods are *connectSession("")* and connectSession(string password). The PsWebsim connectSession method does not allow for multiple named sessions. Since OrgSim design included the ability for user to join a particular session it was necessary to write a method that provided this design requirement. Therefore I wrote the methods *getExistingSession() and checkSession()* (as we saw earlier in section 4.3.1) to connect to a session with a specific label. Besides these two methods I had to write my own Join method in order to be able to re-label players (ie slots) with the simulation in progress.
It should be reminded that Orgsim class extends PsWebsim class and the preliminary initializations like connecting to the Metro Server, is done by PsWebsim class.
The method *getSessionManager()* that is used in the constructor for creating a new game, belongs to the PsWebsim class.

In the next section the interfaces that GameFrame class has implemented will be discussed.

4.3.3.3 The Implemented Interfaces In GameFrame Class

GameFrame class implements four interfaces. These interfaces are PsAdvanceListener, PsJoinListener, PsMessagesListener and PsLeaveListener. The main purposes of these interfaces are to catch the simulation advances, to inform the players if a new player joins or leaves the game and to make it possible for the players to send messages to each other. Let's begin with PsAdvanceListener.

PsAdvanceListener

The main functionality for PsAdvancelistener is to catch the simulation advances. Each time the simulation advances one step, an event is generated by this listener and the method OnPsAdvance() is run. Inside this method we can update the different objects that show the values in which the output variables has taken. When we look inside the method OnPsAdvance() in GameFrame class, we see only three objects are updated. These objects are labels that show the results for

the output variables *Costs, Money and MarketShares*. One question can be raised here is what happens to the rest of the objects in the main window for example the graphs. How they are updated then if they are not updated inside this method. The answer lies inside the method PowersimGUIinit(). This method contains a method call *rememberSeries()*. This method keeps the history for all values taken by a variable. Example of this method is:

model.getPar(37).rememberSeries(1,true);//Total_Demand

Because of this method the update of the graph showing the *Total Demand* output variable, is done automatically without requiring any further code for update.

Refer to OrgSim program to see the details for OnPsAdvance() method and PowersimGUIinit() method.

PsJoinListener

When a player joins a slot (model) this interface generates an event for this joining action and runs the method OnPsJoin(). This method is not only called for the new joined player but also for the other players who have already joined the game. By this way the other players are informed of the new player. I have used this method call to update the list boxes in order to add the name of the new joined player to the list boxes. I have two list boxes that contain the players name, one in Play panel and the other one in the Tools panel and then under TeamResults panel.

Refer to OrgSim program to see the details of this method.

PsMessageListener

This interface can be used for establishing a communication between the players.This interface generates an event when the broadcast() or pointcast() method of the slot (model) is used. Then it calls and runs the method OnPsMessage().

In OrgSim I have used this interface for two purposes. The first one is for sending a message to the other players informing them about the new joined player. This is done by implementing the following code in the constructor for joining an existing game.

model.broadcast("NewPlayerMessage");

It should be noted that this task (informing the other players about the new joined player) can also be carried out in the method OnPsJoin(). But I preferred to use OnPsMessage to do this task. When this method is called a message box appears for the other players telling them about the new joined player.
The second purpose of using the interface PsMessageListener is when the players want to send "Hurry Up" messages to each other. When there are several players in the game, the simulation of the underlying model (OrgModel) relies on all the players having pushed the button "Send". Otherwise the model will not simulate. By using the "Hurry Up" button the players can tell each other to hurry up with playing.
Refer to OrgSim program to see how these tasks are implemented.

PsLeaveListener

The last useful interface that GameFrame class has implemented is PsLeaveListener. When a player quits a game by using the method *Model.leave()*, this interface generates an event for this action and runs the method OnPsLeave(). By this way the slot becomes empty and available to be joined of a new player. The leave() method should be used with extreme care.
Before writing any code here it is worth checking the property for the “Slot Collection” in Powersim Exporter again to be sure that the right property is set (the right property is “Destroy when all leave”).
After setting this property the following code is written for the “Quit” button.

```
model.leave();
session.getSessionManager().close();
setVisible(false);
dispose();
```

The first line of the code empties the slot and makes it available for a new player. The second line cleans the session from the Session Manager it belongs to. It should be noted that this line will have effect only when the last player quits the game otherwise only the slot will be emptied. The third and fourth lines will clean and remove OrgSim window.

Chapter 5 Research On Running OrgSim On The BT Intranet

5.1 Review of the research subject and research approach

After developing the multi-player simulator "OrgSim", the next step of the research is to see how robust this technology (OrgSim) is regarding working on different pc'platforms and browsers. To be specific about the research objectives, the following aspects of the multi-player simulator OrgSim, are desired to be tested:

- The robustness of OrgSim (how stable is OrgSim and does it crash often?);
- How easy it is to start;
- How long it takes to download OrgSim (was it a reasonable period of time to wait?);
- Are there any glitches on their interfaces;
- Does the interface update itself properly when a new player joins the game;
- Does it make the simulation unstable when a new player joins the game;
- Does the repetition of simulation affect the interface;
- What kind of things the players feel that is missing;
- What kind of things the players feel it is unnecessary to have in the interface;
- Do they feel that they have control over the simulation and can understand the process;
- Does it affect the simulation when different players are running different browsers or running on different platforms;
- How does it affect the on going simulation when the fifth player joins the game;

To do this research in a systematic way, I chose the following research approach:

- Trying to run OrgSim on the local machine just by clicking on the html file that embeds OrgSim;
- Setting up a personal web server on my local machine and try to run it from there on both browsers NC and IE;

- Publishing OrgSim on BT web server in order to run it on the Intranet;

It may be reminded that the browsers should be Java enabled browsers that can run Java applets like OrgSim.
The main browser that is used in BT as a standard browser is Netscape Communicator 4.05. In my research I have used the mentioned Netscape browser and Internet Explorer 5.0.
The reason that an applet can run on different browsers and different pc' platforms is because of Java Virtual Machine (JVM) that is integrated with the browsers. In the following section I will present this tool briefly to see how it works.

5.1.1 Java Virtual Machine (JVM)

Traditional compiled programming languages are compiled to machine-level binary code that is, of course, specific to the machine or platform on which it is compiled. The advantage is that the compiled binary runs quickly because it is running in the machine's native language. The disadvantage is that a program written in a traditional language has to be recompiled to binary code for each different hardware platform before it can be run on that machine. On the other hand, whereas traditional interpreted programming languages are machine-neutral and can be run without recompilation on various platforms, they generally run much slower than compiled applications. Java has incorporated the best of both worlds. It has been created a platform-specific layer, called the Java Virtual Machine (Java VM or JVM), which interfaces between the hardware and the Java program.
When a Java-capable browser is installed on the computer, a copy of the Java interpreter (JVM) is also installed, which is what enables the browser to execute Java applets and the system can run standalone Java programs.
In brief, the Java Virtual Machine presents the same interface to any applets that attempt to run on the system, so to applets, all machines look the same. The applet itself is compiled into a form of interpretable code called Java bytecodes. This special form of code can be run on any platform that has the Java Virtual Machine installed.

In the next section we will go through the different research approaches that was mentioned above in section 5.1 and try to solve the raised problems.

5.2 Running OrgSim on the local machine

This approach requires no preparations at all except to run the Powersim Metro Server locally on the computer. Then OrgSim was run with just clicking on the html file "bt.orgsim.Orgsim.html".

The first run was with Internet Explorer 5.0. OrgSim did not run completely. The Java Console reported an error that says "cannot access "127.0.0.1":7777 " (see *approach1 IE. report* in the appendix). It is difficult to make sense of this error message since the machine's IP address where the Metro server was running, is different from the one that PsWebsim was trying to locate.

I tried to run OrgSim on Netscape Communicator but it did not run either. The Java Console for the Netscape had reported a different error than Internet Explorer. The error sounds like this " Class already loaded" (see *approach1 NC. Report* in the appendix).

This error was not specific and hard to follow because it did not refer to a particular class.

It is worth to mention that BT had used this approach before and managed to run some WebSims by this way. For example the single-player WebSim that was developed by BT had run by the above approach.

Since the errors reported by the IE and NC were hard to locate, I decided to proceed with the next approach that is to run a personal web server on the local machine and see whether I get the same problems.

5.3 Running OrgSim on a personal web server on the local machine

This part of the research needs some preliminary preparations. These preparations are installing a personal web server on the local machine, deploying the project

OrgSim with JBuilder and publishing the necessary files in to the personal web server.
I downloaded and installed a personal web server from Microsoft site (the version of the server was not specified).
Because of the experience that I got from the first approach, I decided to divide this research approach in two parts. In the first part I experimented with Internet Explorer 5.0 and in the second part with Netscape Communicator 4.05. The other two preparations are discussed in detail in these two parts.

5.3.1 Experimenting with Internet Explorer 5.0

The two other preparations after installing the personal web server, are to deploy the project OrgSim with JBuilder and to publish the resulting jar file from the deployment to a personal web server. Before continuing the discussion let us see what deployment and publishing are.

What is deployment?

When an applet or an application in general is developed in JBuilder tool or any other tool, all the files needed for distribution of the applet or the application, must be collected and placed in one file. JBuilder has a program that takes care of this task. This program is integrated with JBuilder and is called "Deployment Wizard". Deployment Wizard has some options that can be set by the user. Options like specifying the type (Jar or Zip) of the file that will be generated and will contain all other files (the applet or application files), specifying if the generated file will be compressed or not, which dependency files (JBCL or JGL or both) will be included with the applet or application files, the name of the generated file and where it will be saved.
To see some more details about this tool refer to the documentation for JBuilder.

What is publishing?

To make documents, applications or an applet available to Internet or Intranet visitors, we need to put them in a web server. This action is called publishing. We publish OrgSim (which is an applet) to a personal web server to make it available for the people who want to get access to it via their browsers. In my case I do so in order to run some tests on OrgSim.

Now let’s continue with the research in the following.

I deployed the project with the "JBCL" and "Jar" options checked and used "Orgsim.jar" as the name of the resulting jar file. Then I published the jar file "Orgsim.jar" along with the html file "bt.orgsim.Orgsim.html" in to the personal web server.

When I tried to run OrgSim from the browser, it did not run. The Java Console reported an error that it could not find a class file called "powrsim.Metro.simul.Res_en_GB" (see *approach2-1 IE. Report)*. Actually it does not exist such a class file called Res_en_GB but this should not be a problem due to the way that the getBundle method works for finding localized versions of ResourceBundles. If it can't find localized versions like Res_en_GB (which is English language,UK version) it should try Res_en (English language default) and if it can't find that either it should default to Res. And, powersim.Metro.simul.Res does exist. All this should be standard Java 1.1. Since this did not work properly I got around this problem by importing the file "Res.class" to Orgsim project manually and redeploy the project with the same options.

After publishing the resultant jar file, I browsed OrgSim and it seemed that my solution worked out the problem but Java Console has generated new error.

This time the error was about not finding a file called "pair.class" that is an extension of the JGL class (see *approach2-2 IE. report in appendix)*. To solve this problem I simply redeployed the project with the "JGL" option checked. The resultant jar file was published again. When I browsed OrgSim, it seemed that the problem was worked out and the Java Console has reported a new error.

The next reported error was about a missing file again and it was called "MetalLookAndFeel.class" (see *approach2-3 IE. Report)*. The strange thing that is happening here is that this file extends the swing class and OrgSim does not use the swing class at all. I revised the project Orgsim to check again for the swing class but I didn't find any. Although I did not understand why this error was reported, I managed to sort it out. I solved the problem as follows:

1. I copied a file called swingall.jar from C:\JBuilder2\lib to the personal web server and under the same directory where Orgsim.jar was;
2. I added the name of the jar file to html "bt.orgsim.Orgsim.html" as follows
   ARCHIVE = Orgsim.jar,swingall.jar

When I browsed OrgSim this time, it began to run until the wizard windows showed up. It seemed that the above problem had been sorted out.
The wizard windows worked fine but the main window (the window where the player can play against each other) after the wizard windows did not appear. When I checked the Java Console, new error was reported. It was about the "Res_en_GB" again (see *approach2-4 IE. report in appendix)*. This time the file was missing from "powersim/Look/beans/".
It seemed that either the browser could not locate the file even when it was there in the jar file "Orgsim.jar", or the JBuilder did not deploy the file properly. So to solve this problem I published the powersim folder from C:\WebsimDvl\Java\lib\powersim to the personal web server and under the same directory as the "Orgsim.jar".
I browsed OrgSim and this time it worked and besides the wizard windows, the main window was shown too.

5.3.2 The interface glitches in IE 5.0

OrgSim interface that appeared on the browser on my computer was quite good except some minor glitches. These glitches were as follows.
The look of the first wizard window was ok except that the radio buttons did not update properly. For example when I clicked on one of the radio buttons should the other one turned itself off but it didn't. This was not an unexpected glitch since I had this problem during the development of the Interface in JBuilder. For some reasons the JBuilder could not update the radio buttons properly. It should be a way to get around this problem but since it was not in the core of the WebSim I chose to concentrate on the other issues of the research. The two other wizard windows were ok with no glitches.
When the main window for the WebSim appears, the tab panel "Play" is the first one that is shown. The objects in this tab panel were not in their right places as they should. It seemed that they have been pushed down in the panel. The look for the "Tools" tab panel was ok. Inside the "Tools" tab panel there are three other tab panels. It is just the "TeamResults" tab panel that has some objects and the other two are empty and under construction. The first look of the "TeamResults" tab panel was ok except when the drop down list boxes were activated. They became very funny and strange.

The other tab panels had quite satisfying interface and they did not have glitches in their look.
One possible reason that I could think for these glitches was that the JVM (Java Virtual Machine) for the browser was not operating properly.
BT in its research and development of a single-user WebSim, had problem with Res_en_GB class. The problem was sorted out by downloading the latest version of JVM and the same solution was suggested here for this case.
I downloaded the latest version of the JVM **(**version 3.2**)** and installed on my computer. When I restarted the computer and the browser, all the above mentioned glitches were gone except the update problem with the radio buttons. I ran one round of simulation and the interface did not show any glitches at all.

5.3.3 Experimenting with the Netscape Communicator 4.05

The Netscape Communicator is the standard browser used in BT. Therefore it seemed quite important to get run OrgSim in this browser. To start with I chose an old version of browser (version 4.05) since not everybody was running the latest version of this browser which is NC 4.51.
Before beginning the investigation of running OrgSim on the Netscape Communicator, some preparations had to be done. To separate the files that were used to run OrgSim on the Internet Explorer from the files that will be needed to run OrgSim on the Netscape, I made a new directory on the personal web server. I called this directory Ncorgsim and located it on the following path InetPub\wwwroot\websimroot\Ncorgsim
The next thing to do was to copy the WebsimDvl folder that contained Orgsim project and renamed it to WebsimDvlNC. By this way I had a back up of the project.
I deployed the project with options “include JBCL” and “Jar” checked. I named the jar file “OrgsimNC.jar” and save it under C:\WebsimDvlNC\Java\lib\ .
The jar file and the html file "bt.orgsim.Orgsim.html" were published to the folder Ncorgsim on the personal web server.
I started the Netscape Communicator 4.05 and tried to run the Orgism by pointing to the html file on the personal web server (http://salemim.hermes.bt.co.uk/Murtwebsim/Orgsim/bt.orgsim.Orgsim.html).

OrgSim did not run and Java Console reported two errors (see *approach 2-1 NC report in appendix)*. The first error was a NullPointerException that was generated on the following method:

sun.awt.windows.WComponentPeer.repaint(Compiled Code).

To work out the problem I removed the code "*this.repaint* " from the following methods in Orgsim project:

void tabsetPanel1_mouseClicked(MouseEvent e);

void tabsetPanel2_mouseClicked(MouseEvent e);

When I ran OrgSim in NC, it seemed that the first problem was solved but Java Console was still reporting the second error. The second error was about not to being able to find the resource for the class file Res_en_GB. The file "Res.class" exists under "ProgramFiles\Powersim\Metro\Java API1.1\lib\powersim\Metro\simul\ " and as I mentioned earlier in the previous section this should not be a problem due to the way that the getBundle method works for finding localized versions of ResourceBundles. To sort out this problem I proceeded with the same solution that I used for IE. In brief the solution was to open the file "Res.class" in the project Orgsim and redeploy the project with this new file. The same options (checking the JBCL and jar options) were used.

After copying the resulting jar file to the folder NCorgsim, I tried to run OrgSim from the NC. Unfortunately it did not work and the Java Console reported a new error (see *approach 2-2 NC. report* in the appendix). This error says that a class is already loaded but does not specify which class it is. My guess is that it is referring to the "Res.class" file.

I removed this class file and redeployed the project with the same options and published it to the personal web server. When I ran OrgSim with the new jar file in NC, it reported again the same second error that we had in *approach 2-1 NC. report*.

To go around this problem I thought of another solution. I made a copy of the file "Res.class" and renamed it to "Res_en_GB.class" and put it in the same directory as "Res.class" file (WebsimDvl\Java\lib\powersim\Metro\simul\).

When I ran OrgSim, the Java Console reported a new error that says "Wrong class name inside class file" (see *approach 2-3 NC. report in appendix).*

As this error was persistent and time consuming to solve I downloaded and installed the latest version of the Netscape Communicator (version 4.51) and tested OrgSim with this browser.
When I browsed OrgSim from the personal web server, it ran just fine with no problem and Java Console did not report any errors at all.

5.3.4 The interface glitches in NC 4.51

As with the IE 5.0, the look of the first wizard window was ok except the updating problem that radio buttons had. The second wizard window was ok too but the third wizard window had a minor glitch with the label that says “There is no session running at the moment”. This label was not shown clearly.
The main window had no glitches and all tab panels were working properly with their objects placed in their right positions.

5.3.5 Running OrgSim from other computers

Until now all the tests have been run on my computer. The next test that was more important and exciting to do was to let other machines browse OrgSim from the personal web server on my computer.
In the first trial just one machine was used to browse OrgSim. The machine was a lap top computer with Windows95 platform. The browser NC 4.08 (older than NC 4.51) was used. It browsed OrgSim from the personal web server that was located on my computer. OrgSim ran smoothly without any problem in this machine.
After a few simulations I joined the game from my computer and we began to play against each other. The simulator worked properly on both browsers (I used IE 5.0 browser).
In the next trial I started a new game and three players joined the game from their own computers. All the computers were running Windows95 with different browsers. The IE browsers 4.01, 5.0 and NC browsers 4.5 and 4.08 were used.
The simulation ran almost correctly on the four computers except some anomalies occurred with updating the number of months. It was not updated correctly when a new player joined the game. It did not increment correctly either. These problems did not seem to be a browser defect but rather Java programming

problem. There was also a minor problem with updating the name for the joined player in the drop down list box that was on the “TeamResults” tab panel.
I fixed all these problems in JBuilder and OrgSim is working now properly as it is expected.

5.4 Running OrgSim on the BT Intranet

The next and last step of this research is to run OrgSim on the Intranet.
This requires to put OrgSim on of the BT web severs (pegasus web server).
Pegasus web server runs the Metro server too.
The main purpose by carrying out the previous research was kind of preparation for this important research. It is here that OrgSim should prove that it is working properly on different browsers and platforms. The main reason for this importance, is that this would be actual operational configuration in which OrgSim would be run for real.
It should be noticed that the version of the browsers that will be used, must be the same as those were used earlier in the research.
These browsers were IE 4.01 and NC 4.08 or later versions of these two browsers.

To conduct a systematic test I set up the following approach:

1. Running OrgSim with just one player;
2. Running OrgSim with several players where the players existed on the same LAN;
3. Running OrgSim between Martlesham and Bath sites, the latter connected to the Martlesham LAN via an ISDN connection;

In all these three approaches the results of the tests were almost the same.
The only thing that the two first approaches differed from the third one was the downloading time (caused by the slower ISDN link in the third configuration).
Therefore in the following I will just elaborate on the results obtained for the third test.
In section 5.1 it was mentioned some specific aspects of OrgSim that were desired to be tested. In the following we will see the results of the tests for these aspects.

- How easy it is to start OrgSim;

OrgSim does not require any preliminary preparations to start except Java enabled browsers like IE (version 4.1 or later) or NC (version 4.08 or later) and the URL address to where OrgSim is placed. When the user opens the html page that contains OrgSim, the applet starts automatically. This always worked without fail.

- How long it takes to download OrgSim (is it a reasonable period of time to wait?);
  OrgSim is about 6 MB. The files that are needed to run OrgSim are as follows:
  - A powersim folder that is about 396 KB;
  - Html file (bt.orgsim.Orgsim.html) about 3 KB;
  - An image file about 12 KB;
  - Orgsim jar file about 692 KB;
  - Swingall jar file about 5390 KB;

  Since my machine was on the same LAN as the web server that contains OrgSim was (BT labs in Martlesham-Ipswich), it did not take long time for the browser (IE 5.0) to download OrgSim (about 1-2 minutes). But the other machine that was placed in Bath and was using ISDN line, it took about 10-15 minutes for the browser (NC 4.08) to download OrgSim. This amount of time seems to be long for a user and not quite reasonable. It should be noted that subsequent load are much faster since the applet gets cached.

- The robustness of OrgSim (how stable is OrgSim and does it crash often?);
  OrgSim was run several times and most of the times it run well.
  The reason for crashing sometimes was not always obvious but one possible reason could be because of the way the codes were written. For example when the objects like clear button on the “Tools” tab panel was tried, sometimes the session crashed and sometimes not. The other tab panels in the interface seemed to work properly and did not have effect on the simulation.

- Are there any glitches on the interfaces;
  Some interface glitches were detected. These glitches are as follows:

The radio buttons on the first screen of the wizard Window do not update themselves properly. I was aware of this glitch during the design of OrgSim and it seemed that it was a JBuilder bug rather than my code implementation of the radio buttons.

The next glitch was on the time-table object that was under “log” tab panel. The time-table object did not function correctly all the time. Sometimes it showed only 5 columns instead of 6 columns (as it should) .The information inside these columns which is numbers were not placed in their right columns and it seemed that they were pushed one column to the left. When a number was big, it seemed that not whole the number was shown due to the small size of the cell that contained the number. I did not manage to find the reason for this problem. Time-table is a Powersim class and it could be that this class shows this undesired feature sometimes.

The other undesired feature but not necessarily a glitch, was the way the tab panels updated themselves. Actually the tab panels work properly but the way that they update themselves each time they are clicked, is not quite effective or desirable. The whole screen for the interface flashes each time a tab panel is clicked. During the design time of OrgSim, I observed that the tab panels did not update themselves properly. This also seemed to be another bug in JBuilder that could not update the tab panels in a proper way. To get around this problem, I wrote code that updated the whole window in order to update the tab panels. The effect of this implementation as it is mentioned, is that the whole screen of the interface flashes each time a tab panel is clicked.

The next glitch has to do with the *clear* button found on “Tools” panel. This button sometimes works and sometimes not. The cause of this problem seems to be with JBuilder that does not execute properly the code written for this button.

Another observed undesired feature was when a new player joined a game, a message box appeared for the other player, telling him that a new player was joining the game. This message box did appear to the other player but if this player did not have his main window (play window) active then he could not see this message until he made the main window active. The cause of this problem most likely lies in the code implementation that can be improved.

The last glitch was that when a fifth player wanted to join an existing running game, a message box should have appeared for him, telling him that the session is full. Then he would be sent back to the wizard window to either join an another existing game or create a new game. But the message box did not appear and the wizard window disappeared and nothing happened. This had most likely to do with the code implementation that can be easily corrected.

- Does the interface show any glitches during the simulation;
  In one test the graph for Costs&Revenues had a funny look. The labels were written twice and the graph became an area graph. Otherwise it functioned properly in the other tests.

- Does the interface update itself properly when a new player joins the game;
  The interface for the already existed player updated itself properly. But the interface for the joining player did not always updated itself correctly. It seemed that when the new player joined the game while the simulation was not running, then the interface for the joining player updated itself properly. Otherwise if he joined the game while the simulation was running then the months label did not show the right month on the play screen. As a consequence for this glitch, the two players were not synchronized completely. This glitch did not stop the simulation but just the players entered their decisions in different months.

- Does it make the simulation unstable when a new player joins the game;
  No, it did not affect the stability of OrgSim when a new player joined the game.

- Does the repetition of simulation affect the interface;
  No, the repetition of simulation did not produced any additional glitches on the interface.

- Does it affect the simulation when different players are running different browsers or running on different platforms;

No, it did not seem running different browsers or different platforms had effect what so ever on the simulation.

- How does it affect the on going simulation when the fifth player wants to join the game;
  It did not affect the running simulation at all.

In the following section I will do some assessments based on the above tests.

5.5 Assessments Of Running OrgSim On Intranet

From a general point of view OrgSim works. But let's go through the different aspects tested above and see what actually have been achieved.

- Concerning running OrgSim on different platforms and browsers, the goal had been achieved. OrgSim runs with both browsers IE and NC. It is easy to start and requires no further preparations for running.
- Regarding the downloading time for OrgSim, it is not quite satisfying. As we saw above in previous section the player who had ISDN line, took him about 10-15 minutes to download OrgSim. It is certainly possible to optimize OrgSim by shrinking the jar files "swingall.jar" (5390 KB) and "Orgsim.jar" (692 KB) and sort out the classes that are not needed by OrgSim.
- Concerning OrgSim's robustness and reliability, it can be said that this goal have almost been achieved but not completely. Some further research should be done to find out why OrgSim sometimes for no reason just stops simulating.
  One theory that I have is the way tab panels update themselves may have some negative effects on the stability of the simulation. This also concerns using the tools found on "Tools" tab panel. For example using the clear button or the list boxes for showing the results of the other players, had sometimes and not always some negative effects on the simulation and the session.
- Regarding the glitches found on the interface, it requires some further research to find out the real causes for these glitches. Glitches like those found in radio buttons did not cause serious problem for OrgSim but a glitch in the time-table object can confuse the player. In general if OrgSim is used carefully, these

glitches can be avoided for the moment until some solutions are found for these problems.

- Concerning the behavior of OrgSim when a new player joins an existing game, it can be said that for some extent of degree, OrgSim works well but there are some undesired features with the synchronization of the players. The cause of this problem lies in the way synchronization is implemented and can be improved.

To summarize the assessments it can be concluded that OrgSim works and for up to a reasonable extent of degree, it is robust. But improvements can be done to enhance its functionality, robustness and reliability.

Chapter 6 Conclusion

6.1 The research achievements

Viewing the research from its objectives, it seems that it has reached its goals. OrgSim as a multi-player simulator is built and developed using Powersim Metro and JBuilder tool. In OrgSim up to 4 players can participate in the game and compete with each other. The players can enter their decisions into two input variables and run the simulation to see the results of the their decisions. The results are mainly shown by time-graphs or time-table which are objects of Powersim Metro.
The players can compare their results with each other through the implemented tools in OrgSim.
The development process of OrgSim is documented in detail.
Research on robustness and reliability of this technology is done and different aspects of OrgSim are tested and documented.

The main challenges experienced in this research were to develop the multi-player simulator and make it work on different browsers and platforms. During these two phases of development and implementation, it was experienced many design and implementation problems. Due to the limited time set for this research, it was not possible to handle all the problems. Therefore there are still some aspects of OrgSim that need further research or improvements.
In the following section I present in brief what can be done to improve OrgSim.

6.2 The current status of OrgSim and where to go from here

Generally OrgSim works and four players can compete against each other in this multi-player simulator. But there are undesired features and bugs in OrgSim that should be improved and enhanced. The problems may be caused by bugs in OrgSim code but I haven't been able to rule out that some of them are due to JBuilder, Powersim or other tools used (browsers). Some of the adapted solutions are not quite clear in how they can solve the problems and some further investigations are required to find out the answers.

The features that could be improved and enhanced are as follows:

- The synchronization between the players when a new player joins the game;
- Making the play screen (see figure 4.13) active when the simulation is running so the player can see that there is something happening (at the moment nothing special happens on the play screen when the player pushes the send button. Therefor the play screen seems to be dead during the simulation);
- Hurry-up message box is shown to the other player when he has his play window active. If he has some other windows active instead of the play window and another player sends a hurry-up message to this player, he will not see it before he makes his play window active again;
- Nothing to prevent two players from choosing the same name;
- The way tab panels update themselves can be improved;
- The functionality of clear button in "Tools" can be improved;
- Some help information can be written for OrgSim;

The Bugs found in OrgSim are as follows:

- Time-table in log screen sometimes shows 5 column instead of 6 column. when 5 columns are shown, the numbers under each column are pushed one column to the left and thereby they do not stand under their right column;
- Number format in tables (can't see whole number)
- The radio buttons in the first screen for the wizard window do not update correctly when they are clicked for the first time;
- The message box for the fifth player which supposed to appear for him when he tries to join an existing game, does not come up;

Appendix

Approach 1 IE. report

The following is system messages that fired by Metro Server and shown in Java Console. This error report relates to section 5.2 when trying to run OrgSim in the local machine by clicking on the html file (the Metro Server was running on the local machine).

com.ms.security.SecurityExceptionEx[powersim/Metro/server/PsServer.<init>]: cannot access "127.0.0.1":7777

at com/ms/security/permissions/NetIOPermission.check
at com/ms/security/PolicyEngine.deepCheck
at com/ms/security/PolicyEngine.checkPermission
at com/ms/security/StandardSecurityManager.chk
at com/ms/security/StandardSecurityManager.chkex
at com/ms/security/StandardSecurityManager.checkConnect
at java/net/Socket.<init>
at java/net/Socket.<init>
at powersim/Metro/server/PsServer.<init>
at powersim/Metro/lang/PsSessionManager.<init>
at powersim/Metro/simul/PsWebsim.createSessionManager
at powersim/Metro/simul/PsWebsim.init
at bt/orgsim/Orgsim.init
at com/ms/applet/AppletPanel.securedCall0
at com/ms/applet/AppletPanel.securedCall
at com/ms/applet/AppletPanel.processSentEvent
at com/ms/applet/AppletPanel.processSentEvent
at com/ms/applet/AppletPanel.run
at java/lang/Thread.run

approach 1 NC. report

The following is system messages that fired by JVM for NC and shown in Java Console. This error report relates to section 5.2 when trying to run OrgSim in the local machine by clicking on the html file (the Metro Server was running on the local machine).

Netscape Communications Corporation -- Java 1.1.2
Type '?' for options.
Symantec Java! ByteCode Compiler Version 210.065
Copyright (C) 1996-97 Symantec Corporation

 # Applet exception: error: java.lang.ClassFormatError: Class already loaded

java.lang.ClassFormatError: Class already loaded

at java.lang.ClassLoader.defineClass(Compiled Code)

at netscape.applet.AppletClassLoader.findClass(Compiled Code)

at netscape.applet.AppletClassLoader.loadClass1(Compiled Code)

at netscape.applet.AppletClassLoader.loadClass(Compiled Code)

** at java.lang.ClassLoader.loadClassInternal(Compiled Code)*

at bt.orgsim.Orgsim.<init>(Compiled Code)

at netscape.applet.DerivedAppletFrame.run(Compiled Code)

at java.lang.Thread.run(Compiled Code)

approach 2-1 IE report

The following is system messages that fired by JVM for IE and shown in Java Console. This error report relates to section 5.3.1. OrgSim jar file was put on the personal web server and OrgSim was browsed by IE.

java.io.FileNotFoundException:

salemim.hermes.bt.co.uk:80//Murtwebsim/Orgsim/powersim/Metro/simul/Res_en_GB.class

java.io.FileNotFoundException:

salemim.hermes.bt.co.uk:80//Murtwebsim/Orgsim/powersim/Metro/simul/Res_en.class

java.io.FileNotFoundException:

salemim.hermes.bt.co.uk:80//Murtwebsim/Orgsim/powersim/Metro/simul/Res_en_GB.class

approach 2-2 IE. report

The following is system messages that fired by JVM for IE and shown in Java Console. This error report relates to section 5.3.1. OrgSim jar file was put on the personal web server and OrgSim was browsed by IE.

salemim.hermes.bt.co.uk:80//Murtwebsim/Orgsim/com/objectspace/jgl/Pair.class

java.lang.NoClassDefFoundError: com/objectspace/jgl/Pair

at borland/jbcl/control/ButtonControl.<init>

at bt/orgsim/wizards/MainWizPanel.<init>

at bt/orgsim/wizards/wizardFrame.jbInit

at bt/orgsim/wizards/wizardFrame.<init>

at bt/orgsim/Orgsim.jbInit

at bt/orgsim/Orgsim.init

at com/ms/applet/AppletPanel.securedCall0

at com/ms/applet/AppletPanel.securedCall

at com/ms/applet/AppletPanel.processSentEvent

at com/ms/applet/AppletPanel.processSentEvent

at com/ms/applet/AppletPanel.run

at java/lang/Thread.run

approach 2-3 IE. Report

The following is system messages that fired by JVM for IE and shown in Java Console. This error report relates to section 5.3.1. OrgSim jar file was put on the personal web server and OrgSim was browsed by IE.

java.io.FileNotFoundException:
salemim.hermes.bt.co.uk:80//Murtwebsim/Orgsim/com/sun/java/swing/plaf/metal/MetalLookAndFeel.class
java.io.FileNotFoundException:
salemim.hermes.bt.co.uk:80//Murtwebsim/Orgsim/com/sun/java/swing/plaf/metal/MetalLookAndFeel.class
java.lang.Error: can't load com.sun.java.swing.plaf.metal.MetalLookAndFeel
at com/sun/java/swing/UIManager.initializeDefaultLAF
at com/sun/java/swing/UIManager.initialize
at com/sun/java/swing/UIManager.maybeInitialize
at com/sun/java/swing/UIManager.getDefaults
at com/sun/java/swing/UIManager.getColor
at borland/jbcl/view/ButtonView.<init>
at borland/jbcl/control/ButtonControl.<init>
at bt/orgsim/wizards/MainWizPanel.<init>
at bt/orgsim/wizards/wizardFrame.jbInit
at bt/orgsim/wizards/wizardFrame.<init>
at bt/orgsim/Orgsim.jbInit
at bt/orgsim/Orgsim.init
at com/ms/applet/AppletPanel.securedCall0
at com/ms/applet/AppletPanel.securedCall
at com/ms/applet/AppletPanel.processSentEvent
at com/ms/applet/AppletPanel.processSentEvent
at com/ms/applet/AppletPanel.run
at java/lang/Thread.run

approach 2-4 IE. report

The following is system messages that fired by JVM for IE and shown in Java Console. This error report relates to section 5.3.1. OrgSim jar file was put on the personal web server and
OrgSim was browsed by IE.

OrgSim was browsed by IE.
java.io.FileNotFoundException:
salemim.hermes.bt.co.uk:80//Murtwebsim/Orgsim/powersim/Metro/simul/Res_en_GB.class
java.io.FileNotFoundException:
salemim.hermes.bt.co.uk:80//Murtwebsim/Orgsim/powersim/Metro/simul/Res_en.class

java.io.FileNotFoundException:

salemim.hermes.bt.co.uk:80//Murtwebsim/Orgsim/powersim/Metro/simul/Res_en_GB.class

java.io.FileNotFoundException:

salemim.hermes.bt.co.uk:80//Murtwebsim/Orgsim/powersim/Metro/simul/Res_en.class

com.ms.security.SecurityExceptionEx[com/sun/java/swing/Timer.post]: Event queue access denied.

at com/ms/security/permissions/UIPermission.check

at com/ms/security/PolicyEngine.deepCheck

at com/ms/security/PolicyEngine.checkPermission

at com/ms/security/StandardSecurityManager.chk

at com/ms/security/StandardSecurityManager.checkAwtEventQueueAccess

at java/awt/Toolkit.getSystemEventQueue

at com/sun/java/swing/Timer.post

at com/sun/java/swing/TimerQueue.postExpiredTimers

at com/sun/java/swing/TimerQueue.run

at java/lang/Thread.run

java.io.FileNotFoundException:

salemim.hermes.bt.co.uk:80//Murtwebsim/Orgsim/powersim/Look/beans/Res_en_GB.class

java.io.FileNotFoundException:

salemim.hermes.bt.co.uk:80//Murtwebsim/Orgsim/powersim/Look/beans/Res_en.class

java.io.FileNotFoundException:

salemim.hermes.bt.co.uk:80//Murtwebsim/Orgsim/powersim/Look/beans/Res_en_GB.class

java.io.FileNotFoundException:

salemim.hermes.bt.co.uk:80//Murtwebsim/Orgsim/powersim/Look/beans/Res_en.class

java.io.FileNotFoundException:

salemim.hermes.bt.co.uk:80//Murtwebsim/Orgsim/powersim/Look/beans/Res.class

Exception occurred during event dispatching:

java.util.MissingResourceException: can't find resource for powersim.Look.beans.Res_en_GB

at java/util/ResourceBundle.getBundle

at java/util/ResourceBundle.getBundle

at powersim/Look/beans/PsControl.<init>

at powersim/Look/beans/PsTimeSeries.<init>

at bt/orgsim/Game/GameFrame.<init>

at bt/orgsim/Game/GameFrame.<init>

at bt/orgsim/Orgsim.mainGameStarter

at bt/orgsim/wizards/MainWizPanel.btNext_actionPerformed

at bt/orgsim/wizards/MainWizPanel$3.actionPerformed

at borland/jbcl/util/ActionMulticaster.dispatch

at borland/jbcl/view/BeanPanel.processActionEvent

at borland/jbcl/view/ButtonView.processMouseReleased

at borland/jbcl/view/BeanPanel.processMouseEvent

at java/awt/Component.processEvent

at java/awt/Container.processEvent

at borland/jbcl/view/BeanPanel.processEvent

at java/awt/Component.dispatchEventImpl

at java/awt/Container.dispatchEventImpl

at java/awt/Component.dispatchEvent

at java/awt/LightweightDispatcher.retargetMouseEvent

at java/awt/LightweightDispatcher.processMouseEvent

at java/awt/LightweightDispatcher.dispatchEvent

at java/awt/Container.dispatchEventImpl

at java/awt/Component.dispatchEvent

at java/awt/EventDispatchThread.run

approach 2-1 NC report

The following is system messages that fired by JVM for NC and shown in Java Console. This error report relates to section 5.3.3. OrgSim jar file was put on the personal web server and
OrgSim was browsed by NC.

Netscape Communications Corporation -- Java 1.1.2

Type '?' for options.

java.lang.NullPointerException

at sun.awt.windows.WComponentPeer.repaint(Compiled Code)

Symantec Java! ByteCode Compiler Version 210.065

** at java.awt.Component.repaint(Compiled Code)*

at java.awt.Component.repaint(Compiled Code)

Copyright (C) 1996-97 Symantec Corporation

at sun.awt.windows.WFramePeer.handleResize(Compiled Code)

at sun.awt.windows.WToolkit.run(Compiled Code)

at java.lang.Thread.run(Compiled Code)

 # Applet exception: exception: java.util.MissingResourceException: can't find resource for powersim.Metro.simul.Res_en_GB

java.util.MissingResourceException: can't find resource for powersim.Metro.simul.Res_en_GB

at java.lang.Throwable.<init>(Compiled Code)

at java.lang.Exception.<init>(Compiled Code)

at java.lang.RuntimeException.<init>(Compiled Code)

at java.util.MissingResourceException.<init>(Compiled Code)

at java.util.ResourceBundle.getBundle(Compiled Code)

at java.util.ResourceBundle.getBundle(Compiled Code)

** at powersim.Metro.simul.PsWebsim.<init>(Compiled Code)*

at bt.orgsim.Orgsim.<init>(Compiled Code)

at netscape.applet.DerivedAppletFrame.run(Compiled Code)

at java.lang.Thread.run(Compiled Code)

approach 2-2 NC. report

The following is system messages that fired by JVM for NC and shown in Java Console. This error report relates to section 5.3.3. OrgSim jar file was put on the personal web server and
OrgSim was browsed by NC.

The following messages are fired by JVM.
 # Applet exception: error: java.lang.ClassFormatError: Class already loaded
java.lang.ClassFormatError: Class already loaded
at java.lang.ClassLoader.defineClass(Compiled Code)
at netscape.applet.AppletClassLoader.findClass(Compiled Code)
at netscape.applet.AppletClassLoader.loadClass1(Compiled Code)
at netscape.applet.AppletClassLoader.loadClass(Compiled Code)
** at java.lang.ClassLoader.loadClassInternal(Compiled Code)*
at bt.orgsim.Orgsim.<init>(Compiled Code)
at netscape.applet.DerivedAppletFrame.run(Compiled Code)
at java.lang.Thread.run(Compiled Code)

approach 2-3 NC. report

The following is system messages that fired by JVM for NC and shown in Java Console. This error report relates to section 5.3.3. OrgSim jar file was put on the personal web server and
OrgSim was browsed by NC.

Netscape Communications Corporation -- Java 1.1.2
Type '?' for options.
Symantec Java! ByteCode Compiler Version 210.065
Copyright (C) 1996-97 Symantec Corporation
 # Applet exception: error: java.lang.ClassFormatError: Wrong class name inside class file
java.lang.ClassFormatError: Wrong class name inside class file
at java.lang.ClassLoader.defineClass(Compiled Code)
at netscape.applet.AppletClassLoader.loadClass(Compiled Code)
at netscape.applet.AppletClassLoader.findClass(Compiled Code)
at netscape.applet.AppletClassLoader.loadClass1(Compiled Code)
at netscape.applet.AppletClassLoader.loadClass(Compiled Code)
at netscape.applet.AppletClassLoader.loadClass(Compiled Code)
at java.util.ResourceBundle.findBundle(Compiled Code)
at java.util.ResourceBundle.getBundle(Compiled Code)

at java.util.ResourceBundle.getBundle(Compiled Code)

** at powersim.Metro.simul.PsWebsim.<init>(Compiled Code)*

at bt.orgsim.Orgsim.<init>(Compiled Code)

at netscape.applet.DerivedAppletFrame.run(Compiled Code)

at java.lang.Thread.run(Compiled Code)

Reference

1. K.Jensen, Ian Gallacher ,(An Interactive Telecommunications Business Game for Strategic Exploration), paper presented at the international system dynamics conference, Istanbul.
2. (http://web.mit.edu/afs/athena.mit.edu/org/s/sloan/www/).
3. http://www.powersim.no

Summer Improvement On OrgSim

One improvement has been made on OrgSim that reduced the size of the files from almost 6 MB down to 1,42 MB.

The adjustment was done by removing the swingall.jar file (5390 KB) and adding the following code to the Orgsim.java to provide the necessary classes needed in OrgSim.

```
static {
   try {
     UIManager.setLookAndFeel(new
com.sun.java.swing.plaf.metal.MetalLookAndFeel());
}
   catch (Exception e) {}
  }
```

After adding the above code and deploying Orgsim project, the size of the new "Orgsim.jar" file increased up to 1,049 KB.

In total the size of the files needed to run OrgSim is 1,42 MB.